Winter 2014

Sparse machine learning models in bioinformatics

Yifeng Li
University of Windsor

SPARSE MACHINE LEARNING MODELS IN BIOINFORMATICS

by

YIFENG LI

A Dissertation
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada
2013

SPARSE MACHINE LEARNING MODELS IN BIOINFORMATICS

by

YIFENG LI

APPROVED BY:

F. Wu, External Examiner
Department of Mechanical Engineering, University of Saskatchewan

---

R. Caron, Outside Department Reader
Department of Mathematics and Statistics

R. Gras, Department Reader
School of Computer Science

R. Kent, Department Reader
School of Computer Science

L. Rueda, Co-Advisor
School of Computer Science

A. Ngom, Advisor
School of Computer Science

October 11, 2013

Declaration of Co-Authorship / Previous Publication

I Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows:

1. The decomposition method for sparse coding, covered in Chapter 2, is my work in collaboration with Professor R.J. Caron. The work is described in paper [*Y. Li, R.J. Caron, and A. Ngom, "A decomposition method for large-scale sparse coding in representation learning," IEEE World Congress in Computational Intelligence (WCCI), July 2014, to be submitted.*]. Y. Li proposed the initial idea, implemented it, and wrote the draft of the paper. Y. Li conducted the experiments with the guidance of Professors R. Caron. Dr. R. Caron also evaluated the theoretical soundness of this idea. All authors contributed to the final version of this paper.

2. The hierarchical model, described in Chapter 5, stems from the decision-tree based multi-class pair-wise strategy for linear dimensionality reduction and classification, proposed in [*L. Rueda, B.J. Oommen, and C. Henriquez, "Multi-class pairwise linear dimensionality reduction using heteroscedastic schemes," Pattern Recognition, vol. 43, no. 7, pp. 2456 - 2465, 2010.*]. The difference between the former and the latter is that the former learns the tree structure based on a one-versus-rest scheme, while the latter based on a pair-wise scheme. We applied the hierarchical model in [*I. Rezaeian, Y. Li, M. Crozier, E. Andrechek, A. Ngom, L. Rueda, and L. Porter, "Identifying informative genes for prediction of breast cancer subtypes," IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB), Nice, June, 2013, LNBI 7986, pp. 138-148.*] to identify the gene bio-markers of breast cancer subtypes, under the supervision of Professors A. Ngom, L. Rueda, and L. Porter. In this

collaborative work, all authors contribute to the formulation of the hierarchical model. Furthermore, Y. Li wrote the draft of the introduction, method, biological analysis, and conclusions, I. Rezaeian surveyed the related work, conducted the experiments in Weka and wrote the experimental part. The rest of authors proof-read the paper. In this dissertation, I re-implemented the hierarchical model in MATLAB, and redid the experiments using my own implementations of linear models in the environment of MATLAB.

3. The nearest-border technique, covered in Chapter 5, is the outcome of our joint research with Dr. B.J. Oommen with Carleton University, Ottawa. This preliminary work has been accepted as [*Y. Li, B.J. Oommen, A. Ngom, and L. Rueda, "A new paradigm for pattern classification: Nearest border techniques," 26th Australasian Joint Conference on Artificial Intelligence, New Zealand, Dec. 2013, pp. 441-446.*]. Y. Li devised the nearest-border concepts, implemented them in MATLAB, conducted the experiment, and wrote the draft of the methodology, under the supervision of Professors B.J. Oommen, A. Ngom, and L. Rueda. B.J. Oommen analyzed the experimental results, wrote the introduction and related work parts, and significantly polished the paper.

4. In Chapter 6, the time-series spectral clustering method is a joint work with Dr. A. Ngom, L. Rueda, and N. Subhani, published as [*Y. Li, N. Subhani, A. Ngom, and L. Rueda, "Alignment-based versus variation based transformation methods for clustering microarray time-series data," ACM International Conference On Bioinformatics and Computational Biology (BCB), Niagara Falls, NY, Aug. 2010, pp.53-61.*]. The idea of using spectral clustering was proposed by Professors A. Ngom. Professor A. Ngom also designed the algorithm. Y. Li implemented the methods including spectral clustering, variation-based data transformation, and the new clustering validation index. Y. Li also conducted the computational experiments in MATLAB. N. Subhani, draw the figures and computed the accuracy. A. Ngom wrote the draft of the paper. All the authors contributed the final version of this paper. Professor L. Rueda is the earliest contributor of the multiple alignment theory [*L. Rueda, A. Bari, and A. Ngom, "Clustering time-series gene expression data with unequal time intervals," Springer Trans. on Compu. Systems Biology X, vol. 10, no. 1974, pp. 100-123, 2008.*].

5. In Chapter 7, Professor A. Ngom contributes to the theory and algorithm of extending the MMHC algorithm to MMHO-DBN proposed in [*Y. Li and A. Ngom, "The max-min high-order dynamic Bayesian network learning for identifying gene regula-*

tory networks from time-series microarray data," IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB/SSCI), Singapore, Apr. 2013, pp. 83-90.]. Y. Li implemented the algorithm, and did experiment on artificial data and some real-life data. Y. Li wrote the draft of the above paper, and Professor A. Ngom contributed to the final version. The experiment on real-life data and sensitivity analysis were conducted by Dr. H. Chen and J. Zheng of National Technological University, Singapore, in order to test the performance of our MMHO-DBN method. We are working on a journal paper to be submitted to Journal of Computational Biology in October.

6. In Chapter 8, the main theory and algorithm of the imputation methods proposed in [*Y. Li, A. Ngom, and L. Rueda, "Missing value imputation methods for gene-sample-time microarray data analysis," IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), Montreal, Canada, May 2010, pp.183-189.*], is contributed by Professor A. Ngom. Y. Li did literature survey, implemented the algorithms, conducted experiments, and wrote the first draft of the paper. A. Ngom contributed to the final version of this paper.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

II Declaration of Previous Publication

This thesis includes 18 original papers that have been previously published/submitted for publication in peer reviewed journals and conference proceedings, as follows:

Thesis Chapter	Publication Title/Full Citation	Publication Status
Chapter 2	Y. Li and A. Ngom, "Classification approach based on non-negative least squares," *Neurocomputing*, vol. 118, pp. 41-57, 2013.	published
Chapter 2	Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," *BMC Systems Biology*, vol. 7, no. Suppl 4, pp. S6, 2013.	published
Chapter 2	Y. Li and A. Ngom, "Non-negative least squares methods for the classification of high dimensional biological data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 2, pp. 447-456, 2013.	published

Chapter 2	Y. Li and A. Ngom, "Fast sparse representation approaches for the classification of high-dimensional biological data," *IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, Philadelphia, Oct. 2012, pp. 306-311.	published
Chapter 2	Y. Li and A. Ngom, "A new kernel non-negative matrix factorization and its application in microarray data analysis," *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, San Diego, CA, May 2012, pp. 371-378.	published
Chapter 2	Y. Li and A. Ngom, "Fast kernel sparse representation approaches for classification," *IEEE International Conference on Data Mining (ICDM)*, Brussels, Belgium, Dec. 2012, pp. 966-971.	published
Chapter 2	Y. Li and A. Ngom, "Supervised dictionary learning via non-negative matrix factorization for classification," *International Conference on Machine Learning and Applications (ICMLA)*, Boca Raton, Florida, Dec. 2012, pp. 439-443.	published
Chapter 2	Y. Li and A. Ngom, "Versatile sparse matrix factorization and its applications in high-dimensional biological data analysis," *IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB)*, Nice, June, 2013, LNBI 7986, pp. 91-101.	published
Chapter 2	Y. Li, "Sparse representation for machine learning," *26th Canadian Conference on Artificial Intelligence (AI 2013)*, Regina, May, 2013, LNAI 7884, pp. 352-357.	published
Chapter 2	Y. Li, R.J. Caron, and A. Ngom, "A decomposition method for large-scale sparse coding in representation learning," *IEEE World Congress in Computational Intelligence (WCCI)*, July 2014.	in preparation
Chapter 3	Y. Li and A. Ngom, "The non-negative matrix factorization toolbox for biological data mining," *BMC Source Code for Biology and Medicine*, vol. 8, pp. 10, 2013.	published
Chapter 4	Y. Li and A. Ngom, "Classification of clinical gene-sample-time microarray expression data via tensor decomposition methods," *LNBI/LNCS: Selected Papers of 2010 International Meeting on Computational Intelligence Methods for Bioinfomatics and Biostatistics (CIBB)*, vol. 6685, pp. 275-286, 2011.	published
Chapter 4	Y. Li and A. Ngom, "Mining gene-sample-time microarray data," in L. Rueda ed. *Microarray Image and Data Analysis: Theory and Practice*, CRC Press/Taylor & Francis, Dec. 2013.	in press
Chapter 4	Y. Li and A. Ngom, "Classification approach based on non-negative least squares," *Neurocomputing*, vol. 118, pp. 41-57, 2013.	published

Chapter 5	Y. Li, B.J. Oommen, A. Ngom, and L. Rueda, "A new paradigm for pattern classification: nearest border techniques," *26th Australasian Joint Conference on Artificial Intelligence*, New Zealand, Dec. 2013, pp. 441-446.	in press
Chapter 5	I. Rezaeian, Y. Li, M. Crozier, E. Andrechek, A. Ngom, L. Rueda, and L. Porter, "Identifying informative genes for prediction of breast cancer subtypes," *IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB)*, Nice, June, 2013, LNBI 7986, pp. 138-148.	published
Chapter 6	Y. Li, N. Subhani, A. Ngom, and L. Rueda, "Alignment-based versus variation based transformation methods for clustering microarray time-series data," *ACM International Conference On Bioinformatics and Computational Biology (BCB)*, Niagara Falls, NY, Aug. 2010, pp.53-61.	published
Chapter 7	Y. Li and A. Ngom, "The max-min high-order dynamic Bayesian network learning for identifying gene regulatory networks from time-series microarray data," *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB/SSCI)*, Singapore, Apr. 2013, pp. 83-90.	published
Chapter 8	Y. Li and A. Ngom, "Mining gene-sample-time microarray data," in L. Rueda ed. *Microarray Image and Data Analysis: Theory and Practice*, CRC Press/Taylor & Francis, Dec. 2013.	in press
Chapter 8	Y. Li, A. Ngom, and L. Rueda, "Missing value imputation methods for gene-sample-time microarray data analysis," *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, Montreal, Canada, May 2010, pp.183-189.	published
Chapter 8	Y. Li and A. Ngom, "Classification approach based on non-negative least squares," *Neurocomputing*, vol. 118, pp. 41-57, 2013.	published

I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

The meaning of parsimony is twofold in machine learning: either the structure or (and) the parameter of a model can be sparse. Sparse models have many strengths. First, sparsity is an important regularization principle to reduce model complexity and therefore avoid overfitting. Second, in many fields, for example bioinformatics, many high-dimensional data may be generated by a very few number of hidden factors, thus it is more reasonable to use a proper sparse model than a dense model. Third, a sparse model is often easy to interpret. In this dissertation, we investigate the sparse machine learning models and their applications in high-dimensional biological data analysis. We focus our research on five types of sparse models as follows.

First, sparse representation is a parsimonious principle that a sample can be approximated by a sparse linear combination of basis vectors. We explore existing sparse representation models and propose our own sparse representation methods for high dimensional biological data analysis. We derive different sparse representation models from a Bayesian perspective. Two generic dictionary learning frameworks are proposed. Also, kernel and supervised dictionary learning approaches are devised. Furthermore, we propose fast active-set and decomposition methods for the optimization of sparse coding models.

Second, gene-sample-time data are promising in clinical study, but challenging in computation. We propose sparse tensor decomposition methods and kernel methods for the dimensionality reduction and classification of such data. As the extensions of matrix factorization, tensor decomposition techniques can reduce the dimensionality of the gene-sample-time data dramatically, and the kernel methods can run very efficiently on such data.

Third, we explore two sparse regularized linear models for multi-class problems in bioinformatics. Our first method is called the nearest-border classification technique for data with many classes. Our second method is a hierarchical model. It can simultaneously select features and classify samples. Our experiment, on breast tumor subtyping, shows that this model outperforms the one-versus-all strategy in some cases.

Fourth, we propose to use spectral clustering approaches for clustering microarray time-

series data. The approaches are based on two transformations that have been recently introduced, especially for gene expression time-series data, namely, *alignment*-based and *variation*-based transformations. Both transformations have been devised in order to take into account temporal relationships in the data, and have been shown to increase the ability of a clustering method in detecting co-expressed genes. We investigate the performances of these transformations methods, when combined with spectral clustering on two microarray time-series datasets, and discuss their strengths and weaknesses. Our experiments on two well known real-life datasets show the superiority of the alignment-based over the variation-based transformation for finding meaningful groups of co-expressed genes.

Fifth, we propose the *max-min high-order dynamic Bayesian network* (MMHO-DBN) learning algorithm, in order to reconstruct time-delayed gene regulatory networks. Due to the small sample size of the training data and the power-low nature of gene regulatory networks, the structure of the network is restricted by sparsity. We also apply the qualitative probabilistic networks (QPNs) to interpret the interactions learned. Our experiments on both synthetic and real gene expression time-series data show that, MMHO-DBN can obtain better precision than some existing methods, and perform very fast. The QPN analysis can accurately predict types of influences and synergies.

Additionally, since many high dimensional biological data are subject to missing values, we survey various strategies for learning models from incomplete data. We extend the existing imputation methods, originally for two-way data, to methods for gene-sample-time data. We also propose a pair-wise weighting method for computing kernel matrices from incomplete data. Computational evaluations show that both approaches work very robustly.

Dedication

To my ancestors,
for their ceaseless struggles and immortal inspirations ...

To my parents and wife,
for their years of expectation ...

To my children,
for their better tomorrow ...

Acknowledgements

First of all, I would like to greatly thank my supervisor, Dr. Alioune Ngom, who has consistently supported and guided my thesis research, forged my personalities as a potential young scientist, and constantly encouraged me in both study and life.

I also want to express my gratitude to my co-advisor Dr. Luis Rueda, who shared many works with me, gave me many useful suggestions during my research, and donated many toys to my son.

I want to give many thanks to all members of my doctoral committee: Dr. Fangxiang Wu, Dr. Robin Gras, Dr. Robert Kent, Dr. Richard Caron, and Dr. Sévérien Nkurunziza, in my doctoral committee. They sacrificed a lot of valuable times in reviewing my dissertation, giving me constructive feedbacks, and attending my seminars, comprehensive examination, proposal, and defense.

I would like to acknowledge the helps of my collaborators, Dr. John Oommen, Dr. Richard Caron, Dr. Michael Ochs, Dr. Lisa Porter, Dr. Haifen Zhang, and Dr. Jie Zheng. The joint researches have dramatically widen my knowledge. And I enjoy a lot.

I wish to give a big thank to our department and the Office of Graduate Studies who provided considerate services during my study.

I appreciate the sacrifice and understanding of my parents, my relatives, my wife, and my son. During the period of my doctoral study, I had not visited my parents for four years. My wife has accompanied me for four years, and sacrificed her own career. I feel regret to my son that I have had few time to play with him.

This research has been partially supported by IEEE CIS Walter Karplus Summer Research Grant 2010, Ontario Graduate Scholarship 2011-2013, The Natural Sciences and Engineering Research Council of Canada (NSERC) Grants #RGPIN228117-2011, conference travel grants from IEEE CIBCB, IEEE BIBM, and Canadian AI conferences, and scholarships from The University of Windsor. I appreciate all the financial supports I received.

Contents

Declaration of Co-Authorship / Previous Publication **iii**
I Co-Authorship Declaration iii
II Declaration of Previous Publication v

Abstract **ix**

Dedication **xi**

Acknowledgements **xii**

List of Figures **xix**

List of Tables **xxiv**

List of Abbreviations **xxvii**

1 Introduction **1**
1 Bioinformatics Challenges 1
2 Sparse Machine Learning Models 2
3 My Contributions 4

2 Sparse Representation for High-Dimensional Data Analysis **8**
1 Introduction 8
2 Bayesian Sparse Representation 11
2.1 Sparse Bayesian (Latent) Factor Analysis 11
2.2 Bayesian Sparse Representation 13
3 Bayesian Sparse Coding 15
4 Optimization for Sparse Coding 17
4.1 The Optimizations are Quadratic Programmes and Dimension-Free . 17
4.2 Interior-Point Method 18

4.3 Proximal Method . . . 19
4.4 Active-Set Algorithm for l_1LS . . . 20
4.5 Active-Set Algorithm for NNLS and l_1NNLS . . . 24
4.6 Parallel Active-Set Algorithms . . . 24
4.7 Decomposition Method . . . 27
4.8 Decomposition Method for l_1QP . . . 28
4.9 Decomposition Method for NNQP . . . 31
4.10 Kernel Extensions . . . 32
5 The Performance of Sparse Coding . . . 33
5.1 The Performance of Active-Set Sparse Coding for Classification . . . 33
5.2 The Performance of Decomposition Method . . . 35
6 Bayesian Dictionary Learning . . . 37
6.1 Dictionary Learning Models . . . 39
6.2 A Generic Optimization Framework for Dictionary Learning . . . 40
6.3 Classification Approach Based on Dictionary Learning . . . 41
6.4 Kernel Extensions . . . 43
6.5 Computational Experiments . . . 44
7 Versatile Sparse Matrix Factorization . . . 45
7.1 Versatile Sparse Matrix Factorization Model . . . 46
7.2 Optimization . . . 48
7.3 Computational Experiment . . . 53
7.4 Biological Process Identification . . . 54
8 Supervised Dictionary Learning . . . 55
8.1 Computational Experiments . . . 58
9 Local NNLS Method . . . 58
9.1 Related Work and Insights . . . 59
9.2 Local NNLS Classifier . . . 61
9.3 Repetitive LNNLS Classifier . . . 62
10 Conclusions . . . 70

3 The Non-Negative Matrix Factorization and Sparse Representation Toolboxes 73

1 Introduction . . . 73
2 The Non-Negative Matrix Factorization Toolbox . . . 74
2.1 Background . . . 74
2.2 Implementation . . . 76

2.3 Results and Discussions 82
3 The Sparse Representation Toolbox 93
3.1 Introduction 93
3.2 Implementations 94
3.3 Conclusions 96

4 Tensor Decompositions and Kernel Methods for The Classification of Gene-Sample-Time Data 97
1 Introduction 97
2 Tensor Decomposition Methods for Classification 99
2.1 Related Works 100
2.2 Tensor Decomposition for Feature Extraction 101
3 Kernel Methods for Classification 107
3.1 Related Works 107
3.2 Kernel Sparse Coding Method for Classification 109
3.3 Kernel NMF and Dictionary Learning 111
4 Conclusions 112

5 Sparse Regularized Linear Models for Multi-class Classification 113
1 Introduction 113
2 Nearest Border Technique 114
2.1 Introduction 114
2.2 Problem Formulation 118
2.3 Theory of The Nearest Border (NB) Paradigm 120
2.4 NB Classifiers: The Implementations of The NB Paradigm 121
2.5 Experiments 125
3 A hierarchical Model 131
3.1 Training Phase 131
3.2 Prediction Phase 132
3.3 Characteristics of The Method 133
3.4 An Application to Predict Breast Cancer Subtypes 133
4 The Regularized Linear Models and Kernels Toolbox 134
5 Conclusions 135

6 Spectral Method for Clustering Microarray Time-Series Data 137
1 Introduction 137

2 Alignment-Based Data Transformation 139
3 Variation-Based Data Transformation 144
4 Time-Series Spectral Clustering 146
5 Cluster Validity 148
6 Experimental Results and Analysis 149
7 Conclusions 154

7 High-Order Dynamic Bayesian Network Approaches for Identifying Gene Regulatory Networks 156
1 Introduction 156
2 Bayesian Network 158
2.1 Concepts 158
2.2 Learning the Model Structure 159
2.3 Learning from Complete Data 159
2.4 The Structure Prior 166
2.5 Search Methods 167
2.6 Reconstructing Gene Regulatory Networks by BN 168
3 High-Order DBNs for Gene Regulatory Network Identification 168
3.1 Introduction 168
3.2 Modeling Time-Delayed Regulations with HO-DBNs 169
3.3 Related Works 176
3.4 Max-Min High-Order DBNs 177
3.5 Preliminary Experiment 181
4 Qualitative Probabilistic Networks 184
4.1 Qualitative Influence and Synergy 185
4.2 Related Works 187
4.3 Post-Analysis Using QPN 187
5 Implementation 188
6 New Computational Experiments 189
6.1 On Simulated Data 190
6.2 On Real-Life Data 194
7 Conclusions 200

8 Dealing With Missing Values 202
1 Introduction 202
2 Dealing With Missing Values in Two-Way Data 203

3 Dealing With Missing Values in GST Data . . . 205
4 Pair-Wise Weighting Method . . . 206
4.1 Pair-Wise Weighting is A Local Method . . . 206
4.2 Experiment . . . 207
5 The Extensions of KNNimpute and SVDimpute . . . 209
5.1 Average Methods . . . 209
5.2 k-Nearest Neighbor Methods . . . 210
5.3 k-Nearest Neighbor with Multiple Time-Series Alignment . . . 211
5.4 Singular Value Decomposition . . . 211
5.5 Time-Point Estimation for GST Data . . . 212
5.6 Evaluation of Imputation Methods . . . 213
5.7 Experiment . . . 213
6 Conclusions . . . 220

Appendices **221**

Appendix A An Introduction to Tensor Algebra **221**
1 Notations and Tensor Manipulations . . . 221
2 Tensor Decomposition . . . 223

Appendix B An Introduction to Regularized Linear Models **227**
1 Background . . . 227
2 A Review on Linear Models with Various Regularization and Loss Terms . 231
2.1 Hard-Margin l_2-Regularized Linear Model . . . 231
2.2 Hard-Margin C-SVM: l_2-Regularized Linear Model . . . 234
2.3 Soft-Margin C-SVM: l_2-Regularized Hinge-Loss Linear Model . . . 237
2.4 ν-SVM: Another l_2-Regularized Hinge-Loss Linear Model . . . 239
2.5 l_2-Regularized l_1-Loss Linear Model . . . 242
2.6 ε-SVR: l_2-Regularized ε-Insensitive-Loss Linear Model . . . 245
2.7 ν-SVR: Another l_2-Regularized ε-Insensitive-Loss Linear Model . . . 249
2.8 l_2-Regularized Squared-Loss Linear Model . . . 252
2.9 l_2-Regularized Logistic-Loss Linear Model . . . 254
2.10 Hypersphere One-Class SVM . . . 255
2.11 Hyperplane One-Class SVM . . . 258
3 A Review on Linear-Model Based Feature Selection . . . 260
3.1 l_1-norm SVM: l_1-Regularized Hinge-Loss Linear Model . . . 260

3.2 SVM-Recursive Feature Elimination 262
4 A Review on The Decomposition Methods for SVMs 262
4.1 Decomposition Method for C-SVM 264
4.2 SMO Method for C-SVM 265
4.3 Decomposition Method for ν-SVM 271
4.4 SMO Method for ν-SVM 271
4.5 Decomposition Method for Hypersphere One-Class SVM 273
4.6 SMO Method for Hypersphere One-Class SVM 273

Bibliography **278**

Vita Auctoris **302**

List of Figures

2.1 Accuracies of sparse coding and benchmark approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation. . 35

2.2 Computing time of sparse coding and benchmark methods. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation. 36

2.3 Accuracies of dictionary learning approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation. 45

2.4 Computing time of dictionary learning approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation. 46

2.5 The biological processes identified by our implementations of the standard NMF (a) and VSMF (b), and by Ochs *et al.*'s implementations of the standard NMF (c) and CoGAPS (d) [1]. 56
2.6 Classification performance on six data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. 66
2.7 Crucial difference diagram of the Nemenyi test ($\alpha = 0.05$). 68
2.8 Computing time on six data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. . 69

3.1 Heat map of NMF biclustering result. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix. This is a color figure, thus the readability may be affected if printed in grayscale. 84
3.2 Heat map of NMF clustering result on yeast metabolic cycle time-series data. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix. This is a color figure, thus the readability may be affected if printed in grayscale. 85
3.3 Biological processes discovered by NMF on yeast metabolic cycle time-series data. This is a color figure, thus the readability may be affected if printed in grayscale. 86
3.4 Biological processes discovered by NMF on breast cancer time-series data. This is a color figure, thus the readability may be affected if printed in grayscale. 87
3.5 Mean accuracy and standard deviation results of NMF-based feature extraction on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. 89
3.6 The mean accuracy results of NNLS classifier for different amount of noise on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale. 91
3.7 The mean accuracy results of NNLS classifier for different missing value rates on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale. 92

3.8 The fitted learning curves of NNLS and SVM classifiers on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale. 92
3.9 Nemenyi test comparing 8 classifiers over 13 high dimensional biological data ($\boldsymbol{\alpha = 0.05}$). 93

4.1 A three-way tensor representation (left) and a heatmap representation (right) of a GST data set. The black columns in the heatmap representation correspond to missing time points. 98
4.2 From Tucker3 decomposition (top) to Tucker1 decomposition (bottom). . . 102
4.3 Tensor factorization based feature extraction. 104

5.1 A schematic view which shows the difference between *border* patterns and *prototypes*. 117
5.2 Plots of the synthetic data sets. This is a color figure, thus the readability may be affected if printed in grayscale. 125
5.3 Performance on synthetic data. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. . 127
5.4 The accuracies achieved on three-fold cross-Validation for the 17 real-life data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. 130
5.5 An example of the hierarchical model. 132

6.1 (a) Unaligned profiles $x(t)$ and $y(t)$. (b) Aligned profiles $x(t)$ and $y(t)$, after applying $y(t) \leftarrow y(t) - a_{\min}$. 141
6.2 (a) Yeast phases [2], (b) SCMA clusters , and (c) SCVV clusters for *Saccharomyces cerevisiae*. 150
6.3 (a) SCMA clusters and (b) SCVV clusters for *Saccharomyces cerevisiae* with $K = 10$. 152
6.4 (a) SCMA clusters and (b) SCVV clusters with centroids shown, for *Schizosaccharomyces pombe*. 153

7.1 An example of a Bayesian network representing a joint probability distribution. 158
7.2 A Bayesian network, of which the BDeu and BIC scores are computed. . . . 165
7.3 An example of a second-order HO-DBN (top) and a non-stationary HO-DBN (bottom). 170

7.4 An example of a DBN under the second-order stationary Markov assumption. Left: stationary transition network. Right: the GRN obtained by folding the transition network. 171
7.5 The true network and the predicted networks. 182
7.6 The gene regulatory network learned by MMHO-DBN. 183
7.7 The gene regulatory network learned by DBmcmc. 184
7.8 The gene regulatory network learned by DBN-ZC. 184
7.9 The true network from which simulated data are sampled. 191
7.10 The predicted networks using simulated data. 192
7.11 Effect of parameters on the performance of MMHO-DBN on the simulated data. 193
7.12 The actual *Yeast5* networks and the predicted networks using *Yeast5* data. 195
7.13 The actual *Yeast9* network and the predicted networks using *Yeast9* data. . 196
7.14 Effect of parameters on the performance of MMHO-DBN when applied on *Yeast5on* data. 198
7.15 Effect of parameters on the performance of MMHO-DBN when applied on *Yeast5off* data. 199
7.16 Effect of parameters on the performance of MMHO-DBN when applied on *Yeast9* data. 199

8.1 Classification performance on SRBCT as missing rate increases. This is a color figure, thus the readability may be affected if printed in grayscale. . . 208
8.2 Graphical representation of Nemenyi test over microarray data with missing rates 60% and 70%. 209
8.3 Tensor representation (a) and matrix representation (b) of a gene-sample-time microarray data. 210
8.4 k-NN imputation with multiple-alignment. 212
8.5 Experimental procedure of imputation. 214
8.6 Errors of G*impute (a), S*impute (b), and GS*impute (c) methods, for $g\% =$ 0.1% to 20% and $t = 1$. 216

A.1 The mode-1 (a), mode-2 (b), and mode-3 (c) matricizations of the three-way tensor shown in Figure 4.1. 222
A.2 PARAFAC decomposition. 224
A.3 Tucker3 decomposition. 225

B.1 The schematic explanation of some geometric conceptions. In (b), ε is specifically for the ε-insensitive loss. 231

List of Tables

2.1 Mean computing time (in seconds) of each sample when using 5356 samples as training set. . . . 37
2.2 Mean number of iterations of each sample when using 5356 samples as training set. . . . 38
2.3 Mean prediction accuracies of the l_1QP and NNQP models using 10-fold cross-validation. . . . 38
2.4 The existing NMF and SR models. . . . 47
2.5 Classification performance of VSMF compared to the standard NMF. The time is measure by stopwatch timer (the `tic` and `toc` functions in MATLAB) in seconds. . . . 54
2.6 The performance of MNNLS. . . . 58
2.7 High dimensional microarray data sets. . . . 65
2.8 Minimum data size required to obtain significant accuracy ($\alpha = 0.05$). NNLS-NS method only requires a very small number of training samples in order to obtain significant accuracy. . . . 70

3.1 Algorithms of NMF variants. . . . 77
3.2 NMF-based data mining approaches. . . . 83
3.3 Gene set enrichment analysis using Onto-Express for the factor specific genes identified by NMF. . . . 85
3.4 Methods of sparse representation. . . . 94
3.5 Sparse-representation based machine learning methods. . . . 95

4.1 Classification performance of hidden Markov models on complete IFNβ data. 101
4.2 Classification performance of tensor factorization on complete IFNβ data. . 106
4.3 Comparison of running times on complete IFNβ data. . . . 106
4.4 Comparison of classification performance of SVM on complete IFNβ data. . 109

4.5 Comparison of classification performance of sparse coding on complete IFNβ data. 110
4.6 Comparison of classification performance of kernel dictionary learning on complete IFNβ data. 111

5.1 Summary of our NB methods and the benchmark methods used. 126
5.2 Results of the accuracies achieved by a three-fold cross-validation using the new and benchmark algorithms on the artificial data sets. 126
5.3 The real-life data sets used in our experiments. 128
5.4 Results of the accuracies achieved by a three-fold cross-validation using the new and benchmark algorithms on the real-life data sets. 129
5.5 The accuracy of the hierarchical model. 134
5.6 Current implementations of our RLMK toolbox. 135

6.1 Accuracies of SCMA and SCVV on two data sets. 153

7.1 Example data. 165
7.2 Frequency tables and (conditional) probability tables. 165
7.3 The comparison on simulated data. 181
7.4 The comparison of DBNs. 183
7.5 Current implementations of our probabilistic graphical models toolbox. . . . 189
7.6 Comparison on simulated data. 191
7.7 Comparison on real-life data. 197
7.8 Comparison of MMHO-DBN (first-order) and DBNmcmc on real-life data. . 197
7.9 Validating the time-delays on *Yeast9*. 198

8.1 Microarray data sets. The sample size of each class and the total data size are in the last column. 208
8.2 Statistics for the five methods for 29 different percentages (from 0.1% to 20%) of genes which contain at most 1, 2, 3, 4, or 5 missing values, respectively. There is no time point missing in this case. 215
8.3 Mean NRMS error for G*impute with at most one missing value for each selected time-series and one missing time point. 217
8.4 Mean NRMS error for S*impute with at most one missing value for each selected time-series and one missing time point. 218
8.5 Mean NRMS error for GS*impute with at most one missing value for each selected time-series and one missing time point. 219

B.1 Updates b in different situations. 270
B.2 Update R in various situations. 277

List of Abbreviations

ν-*NBN*: ν-nearest border with normalized distance, 124
k-NN: k-nearest neighbors, 59, 115
l_1LS: l_1-least squares, 14
l_1NNLS: l_1-norm regularized non-negative least squares, 16
1-NN: 1-nearest neighbor, 90
3KNNimpute: three-way k-nearest neighbor imputation, 211
3MAimpute: three-way multiple alignment imputation, 211

ALS: alternating least squares, 224
ANLS: alternating non-negative least squares, 224
ARLSimpute: autoregressive model based imputation, 204

BD: Bayesian decomposition, 75
BDe: Bayesian Dirichlet equivalence, 160
BDeu: Bayesian Dirichlet equivalence uniform, 162
BFRM: Bayesian factor regression model, 12
BI: border identification, 116
BIC: Bayesian information criterion, 160, 164
BN: Bayesian network, 157
BPCAimpute: Bayesian principal component analysis based imputation, 204

CANDECOMP: canonical decomposition, 223
CD: crucial difference, 67, 93
CMOS: classification by moments of order statistics, 118
CNN: condensed nearest neighbor, 116
CoGAPS: coordinated gene activity in pattern sets, 55, 75
CPD: conditional probability distribution, 158
CPT: conditional probability table, 158

DAG: directed acyclic graph, 157
DBmcmc: dynamic Bayesian network learning algorithm based on Markov chain Monte Carlo, 181
DBN-ZC: Zou and Conzen's dynamic Bayesian netowrk learning algorithm, 176
DBN: dynamic Bayesian network, 157
DiscHMMs: discriminative hidden Markov models, 100
DMMP: dynamic max-min parent, 177

ELM: extreme learning machine, 207
EM: expectation maximization, 82, 137, 159, 204
ENN: edited nearest neighbor, 116
ESS: equivalent sample size, 162, 180

FA: factor analysis, 9
FO-DBN: first-order dynamic Bayesian network, 157

GAimpute: gene-average imputation, 209
GenHMMs: generative hidden Markov models, 100
GIST: Gastrointestinal stromal tumor, 55
GKNNimpute: gene k-nearest neighbor imputation, 210
GRN: gene regulatory network, 156
GSAimpute: gene-sample-average imputation, 209
GSEA: gene set enrichment analysis, 84
GSKNNimpute: gene-sample k-nearest neighbor imputation, 210
GSSVDimpute: gene-sample singular value decomposition imputation, 212
GST: gene-sample-time, 97, 202
GSVDimpute: gene singular value decomposition imputation, 212

HDI: highest density interval, 123
HDR: highest density region, 123
HK: hierarchical, 134
HO-DBN: high-order dynamic Bayesian network, 157
HONMF: high-order non-negative matrix factorization, 103, 225
HOOI: high-order orthogonal iterations, 104
HOOI: higher-order orthogonal iterations, 225
HOSVD: high-order singular value decomposition, 104, 225

IBIS: integrated Bayesian inference system, 100

IFNβ: interferon beta, 99
INDAFAC: incomplete data PARAFAC, 205
IRMA: *in vivo* reverse-engineering and modeling assessment, 194

KKT: Karush-Kuhn-Tucker, 21, 232
KNNimpute: k-nearest neighbor imputation, 90
KNNimpute: k-nearest-neighbor imputation, 203
KW-KPCA: kernel principal component analysis-based kernel whitening, 119

LASSO: least absolute shrinkage and selection operator, 14
LDR: linear dimension reduction, 101
LLSimpute: local least squares based imputation, 204
LNNLS: local non-negative least squares, 62
LRC: linear regression classification, 95, 207
LSVM: local support vector machine, 59
LTI: linear time invariant, 107

MA: multiple (time-series) alignment, 211
MAP: maximum a posteriori, 14
MCMC: Markov chain Monte Carlo, 12, 76, 167, 176
MDL: maximum description length, 176
ML: maximum likelihood, 163
MLDR: multilinear dimension reduction, 101
MLE: mixtures of local experts, 61
MMHC-BN: max-min hill-climbing Bayesian network, 168
MMHC: max-min hill-climbing, 157, 177
MMHO-DBN: max-min high-order dynamic Bayesian network, 157, 168
MMPC: max-min parent and children, 177
MNNLS: meta-sample based non-negative least squares, 57
mRMR: minimum redundancy maximum relevance, 262
MS: multiple sclerosis, 99
MSRC: meta-sample based sparse representation classification, 55, 207

NB-HP: nearest border approach based on hyperplane, 124
NB-HS: nearest border approach based on hypersphere, 124
NB: nearest border, 114
NC: nearest centroid, 95

NMF: non-negative matrix factorization, 10
NN: nearest neighbor, 16, 115
NNLS: non-negative least squares, 15
NNQP: non-negative quadratic programming, 18, 79
NRMS: normalized root mean squared, 213
NS-NMF: non-smooth non-negative matrix factorization, 74
NS: nearest subspace, 16
NSPCA: non-negative sparse principal component analysis, 82

OBS: optimal brain surgery, 75
ODE: ordinary differential equation, 156
ortho-NMF: orthogonal non-negative matrix factorization, 81
OS: order statistics, 118
OVA: one-versus-all, 134

PARAFAC-ALS-SI: PARAFAC-alternating least squares with single imputation, 205
PARAFAC: parallel factors, 223
PC: parents and children, 167
PCA: principal component analysis, 9, 88
PGM: probabilistic graphical model, 157
PIN: protein interaction network, 176
PNN: prototypes for nearest neighbor, 116
PR: pattern recognition, 114
PRS: prototype reduction scheme, 116
PSVM: profile support vector machine, 60, 62

QP: quadratic programming, 17, 232
QPN: qualitative probabilistic network, 157, 184

RBF: radial basis function, 33
RLMK: regularized linear models and kernels, 134
RLNNLS: repetitive local non-negative least squares, 64
RNN: reduced nearest neighbor, 116

SAimpute: sample-average imputation, 209
SC: sparse candidate, 167
SCMA: spectral clustering with multiple-alignment transformation, 147

SCVV: spectral clustering with variation-based tranformation, 147
SGPLS: sparse generalized partial least squares, 65
SKNNimpute: sample k-nearest neighbor imputation, 210
SMO: self-organizing map, 137
SMO: sequential minimal optimization, 27, 264
SPLSDA: sparse partial least squares discriminant analysis, 65
SR: sparse representation, 9, 13
SRM: structural risk minimization, 229
SSVDimpute: sample singular value decomposition imputation, 212
STD: standard deviation, 53
SV: support vector, 238
SVDD: support vector domain description, 119, 255
SVDimpute: singular value decomposition based imputation, 204
SVM-RFE: support vector machine recursive feature elimination, 113, 262
SVM: support vector machine, 9, 113, 229
SVR: support vector regression, 242

TMG: text to matrix generator, 75

VC-dimension: Vapnik-Chervonenkis-dimension, 229
VCD: variation-based coexpression detection, 138, 144
VO-DBN: variable-order DBN, 176
VQ: vector quantization, 116
VSMF: versatile sparse matrix factorization, 45, 80

WNMF: weighted non-negative matrix factorization, 204

Chapter 1

Introduction

1 Bioinformatics Challenges

In the area of biological and clinical study, a huge amount of various data, such as genome-wide microarray gene expression data, proteomic mass spectrometry data, array comparative genomic hybridization data, and single nucleotide polymorphisms data, have been being produced. These data provide us systemic information which allows us to conduct genome-wide study, and therefore to reach more precise decisions and conclusions than ever before. Statistical learning and computational intelligence are among the main tools to analyze these data. However, there are many difficulties that preventing an efficient and accurate analysis. For example, there are usually tens of thousands of dimensions in microarray gene expression data, while its sample size is usually very small. With this problem, it may be impossible to estimate the parameters of a model, as the number of sufficient samples needed increases exponentially as the number of dimensions. Second, the high-dimensional data often have many redundant features when only few (maybe hidden) features correspond to the desired study. This might drown useful information. In addition to the small sample size, the noise present in biological data and uncertainty in the target variable often lead to overfitting of some models sensitive to noise and uncertainty. Furthermore, the structural information in the data (for example the temporal information in time-series data) should be carefully taken into account in order to obtain better performance.

With the recent advances in microarray technology, the expression levels of genes with respect to samples can be monitored synchronically over a series of time points. Such microarray data have three types of variables, genes, samples, and time points. Thus, they can be represented by a three-way array data and are termed *gene-sample-time* (GST) microarray data or GST data for short. The dimensionality of such data is much larger

than the two-way array data introduced above. Analyzing GST data can help discover the pathway of many diseases due to genetic disorder, for example multiple sclerosis. Machine learning and data mining approaches are among the main tools to analyzing GST data. First of all, since many existing machine learning methods are derived for two-way data where each sample is represented by a vector, it is very challenging to cluster or classify the three-way GST data where each sample is represented by a matrix. Second, the GST data may suffer from a much more severe issue of missing values, as the values of an entire time point can be missing in a matrix sample. Many missing value imputation methods for two-way data can not be directly applied on GST data.

Reconstructing *gene regulatory networks* (GRNs) is the key to understanding the relations and pathways among thousands of genes and regulatory elements. One effort to reach it is using high-throughput microarray gene time-series data. The rationality is based on the fact that the expression level of the genes at the current time can casually affect the expression level of the genes in the subsequent time. There are many challenges in reconstructing GRNs by learning a machine learning model on microarray gene time-series data. First, the expression levels of thousands of genes can only be sampled at a few time points. If the number of the parameters of a model goes exponentially as the number of genes increases, we confront the curses of dimensionality. Second, the data are usually very noisy, thus we should find a stochastic model to cope with it. Third, feedback regulation, time-delayed interaction, and asynchronous regulation are the characteristics of the gene regulation events. A proper learning method should model all these features.

2 Sparse Machine Learning Models

The meaning of parsimony is twofold in machine learning: either the structure or (and) the parameter of a model can be sparse. Sparse models have many strengthes. First, sparsity is an important regularization principle to reduce model complexity and therefore avoid overfitting. Second, in many fields, for example bioinformatics, many high-dimensional data may be generated by a very few number of hidden factors, thus it is more reasonable to use a proper sparse model than a dense model. Third, a sparse model is often easy to interpret. Therefore, in this dissertation, we investigate the sparse machine learning models and their applications in high-dimensional biological data analysis. We focus our research on five types of sparse models including sparse representation, sparse tensor factorization, sparse linear models, sparse spectral method, and sparse probabilistic graphical models.

Sparse representation is a parsimonious principle that a sample can be approximated

by a sparse linear combination of basis vectors. From this point, we can also view the well-known non-negative matrix factorization as a specific sparse representation model. Given a new sample and basis vectors, learning the sparse coefficient is named sparse coding. Given the training data, learning the basis vectors is termed dictionary learning. Dictionary learning is in essence the sparse matrix factorization. In the next chapter, we shall see that, based on sparse representation, clustering, classification, and dimensionality reduction techniques can be devised for high-dimensional data.

Tensor factorization is the extension of matrix factorization in multi-way data which is also called tensor. Basically matrix is a two-way tensor, and the GST data mentioned above can be represented as a three-way tensor. Therefore, sparse tensor factorization is a suitable tool for analyzing GST data. In my thesis study, I investigate the high-order non-negative factorization and other tensor decomposition models for the dimensionality reduction of GST data.

The parameter of the primal or dual form of the linear models (formulated as $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$) can be regularized to sparse. If the variable of a dual form is sparse, it implies that the model parameter $\boldsymbol{w}$ is a sparse linear combination of the training samples. The training samples corresponding to non-zero coefficients are named support vectors. This can be applied as a sample selection technique for classification or regression. If the variable of a primal form is sparse, that is the model parameter $\boldsymbol{w}$ is sparse, the target is approximated as a sparse linear combination of features. The features corresponding to zero coefficients have no contribution, hence can be removed. This can serve as a feature selection technique for high-dimensional data. In this thesis, I discuss many linear models using different regularization terms and loss functions. Especially, I investigate novel techniques that extend linear models to multi-class data.

The spectral clustering method is a model that takes the similarity matrix as input, and then solves the eigen-decomposition of the graph Laplacian matrix. If the weighted adjacency matrix (calculated based on the similarity matrix) is sparse, then the graph Laplacian matrix is sparse as well. It has been shown that spectral clustering is very efficient, because it is much more efficient to eigen-decompose a sparse matrix than a dense matrix. In my thesis study, I apply the spectral method to cluster microarray time-series data. The alignment-based and variation-based transformations are used to compute the similarity matrix.

In a graphical model with discrete variables, the number of parameters of a local conditional distribution, corresponding to a node with its parents, increases exponentially as the number of the parents increases. Also the gene expression time-series data have only a few

number of time points. Therefore, we have to limit the number of parents for each node in order to learn the model parameters, which results in sparse structures. In my thesis study, I investigate the high-order dynamic Bayesian networks to learn the gene regulatory networks from gene expression time-series data.

3 My Contributions

1. In Chapter 2, we explore sparse representation models for the classification and dimensionality reduction of various high-dimensional biological data. Our contributions in this chapter include:

   (a) We discuss the sparse representation from a Bayesian viewpoint.

   (b) We propose the non-negative least squares based sparse coding model and investigate their classification performance.

   (c) We propose the weighted K-nearest neighbor rule for predicting class labels based on the sparse coefficient obtained by a sparse coding model.

   (d) We propose kernel sparse coding models.

   (e) We propose efficient active-set and decomposition methods for learning the parameter of the sparse coding model.

   (f) We propose two unified frameworks for dictionary learning. Under these frameworks, we can easily extend the linear dictionary learning models to kernel ones.

   (g) We investigate kernel-dictionary-learning based dimensionality reduction method for high-dimensional biological data.

   (h) We propose a supervised dictionary learning model for multi-class data based on sub-dictionary learning.

   (i) We propose a cluster-and-classification approach, which is a local learning method, for classifying complexly distributed data.

2. In Chapter 3, we describe our implementations of the sparse representation models derived in Chapter 2 – *non-negative matrix factorization toolbox* (NMF toolbox) and *sparse representation toolbox* (SR toolbox). The following are our contributions:

   (a) The NMF algorithms, in our NMF toolbox, are relatively complete and implemented in MATLAB.

(b) Our NMF toolbox includes many functionalities for mining biological data, such as clustering, biclustering, feature extraction, feature selection, and classification.

(c) The NMF toolbox also provides additional functions for biological data visualization, such as heat-maps and other visualization tools. They are pretty helpful for interpreting some results. Statistical methods are also included for comparing the performances of multiple methods.

(d) The SR toolbox include all the sparse coding and dictionary learning algorithms mentioned Chapter 2.

(e) Based on the basic level of the SR algorithms, machine learning methods, including classification and dimensionality reduction, are implemented in our SR toolbox.

3. In Chapter 4, we propose sparse tensor decomposition methods and kernel methods for the dimensionality reduction and classification of gene-sample-time data. Our contributions are the following:

   (a) We propose to apply the sparse tensor decomposition method, higher-order non-negative matrix factorization, to reduce the dimensionality of the gene-sample-time data efficiently.

   (b) We apply kernel sparse coding and kernel dictionary learning methods for classifying matrix samples and extract vectorial features, respectively.

4. In Chapter 5, we propose two novel strategies to extend linear models to multi-class data. The first one is an independent modeling, called nearest border method. The second one is a hierarchical model. Our contributions include:

   (a) We propose the novel nearest border paradigm for multi-class classification problems, and implemented it by using a one-class SVM. This philosophy has not been presented before in the literature.

   (b) We propose the hierarchical model for multi-class classification problems. This model is so flexible that any feature selection and classification methods can be embedded in the model. We apply this model for gene selection and prediction of breast tumor subtypes, simultaneously.

   (c) We realize most of the regularized linear models based methods mentioned in this chapter and Appendix B in MATLAB, and assemble them to an open-source toolbox named *regularized linear models and kernels* (RLMK) toolbox.

Additionally, we conduct a thorough review on regularized linear models in Appendix B where our contributions include:

(a) We review the main regularized (sparse) linear models using matrix notations, unlike element-wise notations in most of the existing reviews. The formulations using matrices and vectors are succinct and easy to understand.

(b) Two main feature selection techniques based on SVM are also reviewed in details. These techniques have been applied to gene selection in bioinformatics.

(c) We review the decomposition methods for two-class SVMs, and derive decomposition methods for one-class SVMs.

5. In Chapter 6, we propose to use spectral method to cluster microarray time-series data. We compare the alignment-based and validation-based transformations which are used to measure the similarities between pairs of time-series. The contributions of this chapter are three-fold:

   (a) We apply spectral clustering algorithms to expression time-series analysis.

   (b) We propose new measurements for the quality of the spectral clustering approach.

   (c) we empirically show that when applied to two well-known data sets, the alignment-based transformation yields better clustering results than the variation-based transformation.

6. In Chapter 7, we apply probabilistic graphical models on microarray gene expression time-series data to reconstruct the gene regulatory networks. We have the following contributions:

   (a) We propose the *max-min high-order dynamic Bayesian network* (MMHO-DBN) learning algorithm, in order to reconstruct *time-delayed* gene regulatory networks.

   (b) We apply the qualitative probabilistic networks (QPNs), after obtaining a DBN, to interpret the interactions learned using the concepts of influence and synergy.

   (c) We have implemented the MMHO-DBN and QPNs in MATLAB, and published it online.

7. In Chapter 8, we explore various strategies for learning models from incomplete biological data. Our main contributions include:

(a) We extend the existing imputation methods, originally for two-way data, to methods for gene-sample-time data.

(b) We also propose a pair-wise weighting method for computing kernel matrices from incomplete data.

Chapter 2

Sparse Representation for High-Dimensional Data Analysis

1 Introduction [1]

The studies in biology and medicine have been revolutionarily changed since the inventions of many high-throughput sensory techniques. Using these techniques, the molecular phenomena can be probed with a high resolution. In the virtue of such techniques, we are able to conduct systematic genome-wide analysis. In the last decade, many important results have been achieved by analyzing the high-throughput data, such as microarray gene expression profiles, gene copy numbers profiles, proteomic mass spectrometry data, next-generation sequences, and so on.

On one hand, biologists are enjoying the richness of their data; one another hand, bioinformaticians are being challenged by the issues of high-dimensional data as well as by the complexity of bio-molecular data. Many of the analysis can be formulated as machine learning tasks. First of all, we have to face the cures of high dimensionality, which means that many machine learning models are unable to correctly predict the classes of unknown samples due to the large number of features and the small number of samples in such data. In other words, the machine learning models can be overfitted and therefore have poor capability of generalization. Second, if the learning of a model is sensitive to the dimensionality, the learning procedure could be extremely slow. Third, many of the data are very noisy, therefore the robustness of a model is necessary. Fourth, the high-throughput data exhibit a large variability and redundancy, which make the mining of useful knowledge

[1] This section is based on our publication [3].

difficult. Moreover, the observed data usually do not tell us the key points of the story. We need to discover and interpret the latent factors which drive the observed data.

Many of such analyses are classification problems from the machine learning viewpoint. Therefore in this study, we focus our study on the classification techniques for high-dimensional biological data. The machine learning techniques addressing the challenges above can be categorized into two classes. The first one aims to directly classify the high-dimensional data while keeping a good generalization ability and efficiency in optimization. The most popular method in this class is the *regularized basis-expanded linear model*. The regularization aims to avoid overfitting a model by reducing the model complexity while fitting it. One example is the state-of-the-art *support vector machine* (SVM) [4]. SVM is sparse linear model that uses hinge loss and l_2-norm regularization. It can be kernelized and its result is theoretically sound. Combining different regularization terms and various loss functions, we can have many variants of such linear models [5]. In addition to classification, some of the models can be applied to regression and feature (bio-marker) identification. However, most of the learned linear models are not interpretable, while interpretability is usually the requirement of biological data analysis. Moreover, linear models can not be extended *naturally* to multi-class data, while in bioinformatics a class may be composed of many subtypes. See Appendix B for a through discussion on linear models.

Another technique of tackling with the challenges above is dimensionality reduction which includes feature extraction and feature selection. Principal component analysis (PCA) [6] is one of the traditional feature extraction methods and is widely used in processing high-dimensional biological data. The basis vectors produced by PCA are orthogonal, however many patterns in bioinformatics are not orthogonal at all. The classic *factor analysis* (FA) [7] also has orthogonal constraints on the basis vectors, however its Bayesian treatment does not necessarily produce orthogonal basis vectors. Bayesian factor analysis will be introduced in the next section.

Sparse representation (SR) [8] is a parsimonious principle that a sample can be approximated by a sparse linear combination of basis vectors. Non-orthogonal basis vectors can be learned by SR, and the basis vectors may be allowed to be redundant. SR highlights the parsimony in representation learning [9]. This simple principle has many strengthes that encourage us to explore its usefulness in bioinformatics. First, it is very robust to redundancy, because it only selects few among all of the basis vectors. Second, it is very robust to noise [10]. Furthermore, its basis vectors are non-orthogonal, and sometimes are interpretable due to its sparseness [11]. There are two techniques in SR. First, given a basis matrix, learning the sparse coefficients of a new sample is called *sparse coding*. Second, given

training data, learning the basis vector is called *dictionary learning*. As dictionary learning is, in essence, a sparse matrix factorization technique. *Non-negative matrix factorization* (NMF) [12] can be viewed a specific case of SR. For understanding sparse representation better, we will give the formal mathematical formulation from a Bayesian perspective in the next section.

Contributions: In this chapter, we investigate sparse representation for high-dimensional data analysis comprehensively. We explore various sparse representation models for the classification and dimensionality reduction of high-dimensional biological data. We propose kernel and supervised sparse representation models. Efficient optimization algorithms are devised for them. Our contributions in this chapter include:

1. We discuss the sparse representation from a Bayesian viewpoint.
2. We propose the non-negative least squares based sparse coding model and investigate their classification performance.
3. We propose the weighted K-nearest neighbor rule for predicting class labels based on the sparse coefficient obtained by a sparse coding model.
4. We propose kernel sparse coding models.
5. We propose efficient active-set and decomposition methods for learning the parameter of the sparse coding model.
6. We propose two unified frameworks for dictionary learning. Under these frameworks, we can easily extend the linear dictionary learning models to kernel ones.
7. We investigate kernel-dictionary-learning based dimensionality reduction method for high-dimensional biological data.
8. We propose a supervised dictionary learning model for multi-class data based on sub-dictionary learning.
9. We propose a cluster-and-classification approach, which is a local learning method, for classifying complexly distributed data.

The rest of this chapter is organized as follows. First, we formulate sparse representation from a Bayesian viewpoint in Section 2. Then, we discuss sparse-coding based classification and the corresponding optimization algorithms in Sections 3 and 4. After that, we

investigate dictionary learning based dimension reduction techniques in Section 6. In this section, we present a generic dictionary learning framework which can be easily kernelized. In Section 7, a more general model called versatile sparse matrix factorization is proposed. We then propose a supervised dictionary learning method for multi-class data in Section 8. Finally in Section 9, we propose a novel local classification approach, called clustering-and-classifying, that combines both dictionary learning and sparse coding.

2 Bayesian Sparse Representation [2]

Both (sparse) factor analysis and sparse representation models can be used as dimension reduction techniques. Due to their intuitive similarity, it is necessary to give their definitions for comparison. In this section, we briefly survey the sparse factor analysis and sparse representation in a Bayesian viewpoint. The introduction of sparse representation is helpful to understand the content of the subsequent sections. Hereafter, we use the following notations unless otherwise stated. Suppose the training data is $\boldsymbol{D} \in \mathbb{R}^{m\times n}$ (m is the number of features and n is the number of training instances (samples or observations)), the class information is in the n-dimensional vector $\boldsymbol{c}$. Suppose p new instances are in $\boldsymbol{B} \in \mathbb{R}^{m\times p}$.

2.1 Sparse Bayesian (Latent) Factor Analysis

The advantages of *Bayesian (latent) factor analysis model* [13] over likelihood (latent) factor analysis model are that

1. The knowledge regarding the model parameters from experts and previous investigations can be incorporated through the prior.

2. The values of parameters are refined using the current training observations.

The Bayesian factor analysis model [13] can be formulated as

$$(\boldsymbol{b}|\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{x}, k) = \boldsymbol{\mu} + \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon}, \tag{2.1}$$

where $\boldsymbol{b} \in \mathbb{R}^{m\times 1}$ is an observed multivariate variable, $\boldsymbol{\mu} \in \mathbb{R}^{m\times 1}$ is the population mean, $\boldsymbol{A} \in \mathbb{R}^{m\times k}$ is *latent factor loading matrix*, and $\boldsymbol{x} \in \mathbb{R}^{k\times 1}$ is *latent factor score* ($k \ll m$), and $\boldsymbol{\epsilon} \in \mathbb{R}^{m\times 1}$ is an idiosyncratic *error* term. In Equation (2.1), we list the model parameters,

[2] This section is based on our publication [3].

$\{\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{x}, k\}$, explicitly on the righthand side of "—". This model is restricted by the following constraints or assumptions:

1. The error term is normally distributed with mean $\mathbf{0}$ and covariance $\boldsymbol{\Phi}$: $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \boldsymbol{\Phi})$. $\boldsymbol{\Phi}$ is usually diagonal.

2. The factor score vector is also normally distributed with mean $\mathbf{0}$ and identity covariance $\boldsymbol{R} = \boldsymbol{I}$: $\boldsymbol{x} \sim N(\mathbf{0}, \boldsymbol{R})$; and the factor loading vector is normally distributed: $\boldsymbol{a}_i \sim N(\mathbf{0}, \boldsymbol{\Delta})$ where $\boldsymbol{\Delta}$ is diagonal. Alternatively, the factor loading vectors can be normally distributed with mean $\mathbf{0}$ and identity covariance $\boldsymbol{\Delta} = \boldsymbol{I}$; and the factor score vector is normally distributed with mean $\mathbf{0}$ and diagonal covariance $\boldsymbol{R}$. The benefit of identity covariance either on $\boldsymbol{x}$ or $\boldsymbol{A}$ is that arbitrary scale interchange between $\boldsymbol{A}$ and $\boldsymbol{x}$ due to scale invariance can be avoided.

3. $\boldsymbol{x}$ is independent of $\boldsymbol{\epsilon}$.

For n training instances $\boldsymbol{D}$, we have the likelihood:

$$\begin{aligned} p(\boldsymbol{D}|\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{Y}, \boldsymbol{\Phi}, k) &= \frac{1}{(2\pi)^{\frac{mn}{2}}|\boldsymbol{\Phi}|^{\frac{n}{2}}} e^{-\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{d}_i - \boldsymbol{\mu} - \boldsymbol{A}\boldsymbol{y}_i)^{\mathrm{T}}\boldsymbol{\Phi}^{-1}(\boldsymbol{d}_i - \boldsymbol{\mu} - \boldsymbol{A}\boldsymbol{y}_i)} \\ &= \frac{1}{(2\pi)^{\frac{mn}{2}}|\boldsymbol{\Phi}|^{\frac{n}{2}}} e^{-\frac{1}{2}\mathrm{trace}[(\boldsymbol{D} - \boldsymbol{\mu}\mathbf{1}^{\mathrm{T}} - \boldsymbol{A}\boldsymbol{Y})^{\mathrm{T}}\boldsymbol{\Phi}^{-1}(\boldsymbol{D} - \boldsymbol{\mu}\mathbf{1}^{\mathrm{T}} - \boldsymbol{A}\boldsymbol{Y})]}, \end{aligned} \tag{2.2}$$

where $\mathrm{trace}(\boldsymbol{M})$ is the trace of square matrix $\boldsymbol{M}$.

The variants of Bayesian factor analysis models differ in the decomposition of the joint priors. The simplest one may be $p(\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{Y}) = p(\boldsymbol{\mu})p(\boldsymbol{A})p(\boldsymbol{Y})$. Suppose k is fixed a priori. The posterior therefore becomes

$$p(\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{Y}|\boldsymbol{D}, k) \propto p(\boldsymbol{D}|\boldsymbol{\mu}, \boldsymbol{A}, \boldsymbol{Y}, \boldsymbol{\Phi}, k)p(\boldsymbol{\mu})p(\boldsymbol{A})p(\boldsymbol{Y}). \tag{2.3}$$

The model parameters are usually estimated via *Markov chain Monte Carlo* (MCMC) techniques.

Sparse Bayesian factor analysis model imposes a sparsity-inducing distribution over the factor loading matrix instead of Gaussian distribution. In [14], the following mixture of prior is proposed:

$$p(a_{ij}) = (1 - \pi_{ij})\delta_0(a_{ij}) + \pi_{ij}N(a_{ij}|0, 1), \tag{2.4}$$

where π_{ij} is the probability of a nonzero a_{ij} and $\delta_0(\cdot)$ is the Dirac delta function at 0. Meanwhile, $\boldsymbol{A}$ is constrained using the lower triangular method. *Bayesian factor regression*

model (BFRM) is the combination of Bayesian factor analysis and Bayesian regression [14]. It has been applied in oncogenic pathway studies [7] as a variable selection method.

2.2 Bayesian Sparse Representation

Sparse representation (SR) is a principle that a signal can be approximated by a sparse linear combination of dictionary atoms [15]. The SR model can be formulated as

$$\begin{aligned}(\boldsymbol{b}|\boldsymbol{A}, \boldsymbol{x}, k) &= x_1\boldsymbol{a}_1 + \cdots + x_k\boldsymbol{a}_k + \boldsymbol{\epsilon} \\ &= \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon},\end{aligned} \tag{2.5}$$

where $\boldsymbol{A} = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k]$ is called *dictionary*, $\boldsymbol{a}_i$ is a dictionary atom, $\boldsymbol{x}$ is a sparse *coefficient vector*, and $\boldsymbol{\epsilon}$ is an error term. $\boldsymbol{A}$, $\boldsymbol{x}$, and k are the model parameters. SR model has the following constraints:

1. The error term is Gaussian distributed with mean zero and isotropic covariance, that is $\boldsymbol{\epsilon} \sim N(\boldsymbol{0}, \boldsymbol{\Phi})$ where $\boldsymbol{\Phi} = \phi\boldsymbol{I}$ where ϕ is a positive scalar.

2. The dictionary atoms is usually Gaussian distributed, that is $\boldsymbol{a}_i \sim N(\boldsymbol{0}, \boldsymbol{\Delta})$ where $\boldsymbol{\Delta} = \boldsymbol{I}$. The coefficient vector should follows a sparsity-inducing distribution.

3. $\boldsymbol{x}$ is independent of $\boldsymbol{\epsilon}$.

Through comparing the concepts of Bayesian factor analysis and Bayesian sparse representation, we can find that the main difference between them is that the former applies a sparsity-inducing distribution over the factor loading matrix, while the latter uses a sparsity-inducing distribution on the factor score vector.

Sparse representation involves sparse coding and dictionary learning. Given a new signal $\boldsymbol{b}$ and a dictionary $\boldsymbol{A}$, learning the sparse coefficient $\boldsymbol{x}$ is termed *sparse coding*. It can be statistically formulated as

$$(\boldsymbol{b}|\boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{\epsilon}. \tag{2.6}$$

Suppose the coefficient vector has Laplacian prior with zero mean and isotropic variance, that is $p(\boldsymbol{x}|\boldsymbol{\Gamma}) = L(\boldsymbol{0}, \boldsymbol{\Gamma}) = \frac{1}{(2\gamma)^k} e^{-\frac{\|\boldsymbol{x}\|_1}{\gamma}}$. The likelihood is Gaussian distributed as

$p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi}) = N(\boldsymbol{A}\boldsymbol{x},\boldsymbol{\Phi}) = \frac{1}{(2\pi)^{\frac{m}{2}}\phi^{\frac{m}{2}}} e^{-\frac{1}{2\phi}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2}$. The posterior is thus

$$\begin{aligned} p(\boldsymbol{x}|\boldsymbol{A},\boldsymbol{b},\boldsymbol{\Phi},\boldsymbol{\Gamma}) &= \frac{p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi},\boldsymbol{\Gamma})p(\boldsymbol{x}|\boldsymbol{A},\boldsymbol{\Phi},\boldsymbol{\Gamma})}{p(\boldsymbol{b})} \\ &\propto p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi})p(\boldsymbol{x}|\boldsymbol{\Gamma}). \end{aligned} \tag{2.7}$$

Taking the logarithm and substituting the Laplacian prior and Gaussian likelihood, the above is thus

$$\begin{aligned} L(\boldsymbol{x}) &= \log p(\boldsymbol{b}|\boldsymbol{A},\boldsymbol{x},\boldsymbol{\Phi}) + \log p(\boldsymbol{x}|\boldsymbol{\Gamma}) \\ &= -\frac{1}{2\phi}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 - \frac{\|\boldsymbol{x}\|_1}{\gamma} + c, \end{aligned} \tag{2.8}$$

where c is a constant term. We can see that maximizing the posterior is equivalent to minimizing the following task:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}\|_2^2 + \lambda\|\boldsymbol{x}\|_1, \tag{2.9}$$

where $\lambda = \frac{\phi}{\gamma}$. Hereafter we call Equation (2.9) *l_1-least squares* (l_1LS) sparse coding model. It is known as the l_1-regularized regression model in regularization theory. It coincides with the well-known *LASSO* (least absolute shrinkage and selection operator) model [16], which in fact is a *maximum a posteriori* (MAP) estimation.

Given training data $\boldsymbol{D}$, learning (or estimating) the dictionary $\boldsymbol{A}$, the coefficient vectors $\boldsymbol{Y}$, and the number of dictionary atoms k is called *dictionary learning*. Suppose k is given a priori, and consider the Laplacian prior over $\boldsymbol{Y}$ and the Gaussian prior over $\boldsymbol{A}$, and suppose $p(\boldsymbol{A},\boldsymbol{Y}) = p(\boldsymbol{A})p(\boldsymbol{Y}) = \prod_{i=1}^{k}(p(\boldsymbol{a}_i))\prod_{i=1}^{n}(p(\boldsymbol{y}_i))$. We thus have the prior:

$$p(\boldsymbol{A},\boldsymbol{Y}|\boldsymbol{\Delta},\boldsymbol{\Gamma}) = \frac{1}{(2\pi)^{\frac{k}{2}}} e^{\sum_{i=1}^{k} -\frac{1}{2}\|\boldsymbol{a}_i\|_2^2} \frac{1}{(2\gamma)^{kn}} e^{\sum_{i=1}^{n} -\frac{\|\boldsymbol{y}_i\|_1}{\gamma}} \tag{2.10}$$

The likelihood is

$$p(\boldsymbol{D}|\boldsymbol{A},\boldsymbol{Y},\boldsymbol{\Phi}) = \frac{1}{(2\pi)^{\frac{mn}{2}}\phi^{\frac{mn}{2}}} e^{-\frac{1}{2\phi}\mathrm{trace}(\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2)}. \tag{2.11}$$

The posterior is

$$p(\boldsymbol{A}, \boldsymbol{Y}|\boldsymbol{D}, \boldsymbol{\Delta}, \boldsymbol{\Gamma}, \boldsymbol{\Phi}) = \frac{p(\boldsymbol{D}|\boldsymbol{A}, \boldsymbol{Y}, \boldsymbol{\Delta}, \boldsymbol{\Gamma}, \boldsymbol{\Phi}) p(\boldsymbol{A}, \boldsymbol{Y}|\boldsymbol{\Delta}, \boldsymbol{\Gamma}, \boldsymbol{\Phi})}{p(\boldsymbol{D})} \tag{2.12}$$

$$\propto p(\boldsymbol{D}|\boldsymbol{A}, \boldsymbol{Y}, \boldsymbol{\Phi}) p(\boldsymbol{A}|\boldsymbol{\Delta}) p(\boldsymbol{Y}|\boldsymbol{\Gamma}). \tag{2.13}$$

Ignoring the normalization term, the log-posterior is thus

$$L(\boldsymbol{A}, \boldsymbol{Y}) = -\sum_{i=1}^{n} \frac{1}{2\phi} \|\boldsymbol{d}_i - \boldsymbol{A}\boldsymbol{y}_i\|_2^2 - \sum_{i=1}^{k} \frac{1}{2} \|\boldsymbol{a}_i\|_2^2 - \sum_{i=1}^{n} \frac{\|\boldsymbol{y}_i\|_1}{\gamma} + c. \tag{2.14}$$

Therefore the MAP estimation of dictionary learning task is

$$\min_{\boldsymbol{A}, \boldsymbol{Y}} f(\boldsymbol{A}, \boldsymbol{Y}) = \sum_{i=1}^{n} \frac{1}{2} \|\boldsymbol{d}_i - \boldsymbol{A}\boldsymbol{y}_i\|_2^2 + \sum_{i=1}^{k} \frac{\alpha}{2} \|\boldsymbol{a}_i\|_2^2 + \sum_{i=1}^{n} \lambda \|\boldsymbol{y}_i\|_1, \tag{2.15}$$

where $\alpha = \phi$ and $\lambda = \frac{\phi}{\gamma}$. Equation (2.15) is known as a dictionary learning model based on l_1-regularized least squares.

3 Bayesian Sparse Coding [3]

Since, the l_1LS sparse coding (Equation (2.9)) is a two-sided symmetric model, thus a coefficient can be zero, positive, or negative [19]. In Bioinformatics, l_1LS sparse coding has been applied for the classification of microarray gene expression data in [20]. The main idea is in the following. First, training instances are collected in a dictionary. Then, a new instance is regressed by l_1LS sparse coding. Thus its corresponding sparse coefficient vector is obtained. Next, the regression residual of this instance to each class is computed, and finally this instance is assigned to the class with the minimum residual.

We generalize this methodology in the way that the sparse codes can be obtained by many other regularization methods and constraints. For example, we can pool all training instances in a dictionary (hence $k = n$ and $\boldsymbol{A} = \boldsymbol{D}$), and then learn the non-negative coefficient vectors of a new instance, which is formulated as a one-sided model:

$$\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 \text{ s.t. } \boldsymbol{x} \geq 0. \tag{2.16}$$

We called this model the *non-negative least squares* (NNLS) sparse coding. NNLS has

[3] This section is based on our publications [3], [17], and [18].

two advantages over l_1LS. First, the non-negative coefficient vector is more easily interpretable than coefficient vector of mixed signs, under some circumstances. Second, NNLS is a non-parametric model which is more convenient in practice. From a Bayesian viewpoint, Equation (2.16) is equivalent to the MAP estimation with the same Gaussian error as in Equation (2.5), but with the following discrete prior:

$$p(\boldsymbol{x}) = \begin{cases} 0.5^k & \text{if } \boldsymbol{x} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.17}$$

This non-negative prior implies that, the elements in $\boldsymbol{x}$ are independent, and the probability that $x_i = 0$ is 0.5 and the probability that $x_i > 0$ is 0.5 as well. (That is the probabilities of x_i being either 0 or positive are equal, and the probability of being negative is zero.) Inspired by many sparse NMFs, l_1-regularization can be additionally used to produce more sparse coefficients than NNLS above. The combination of l_1-regularization and non-negativity constraint results in the l_1NNLS sparse coding model as formulated below:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{x}, \text{ s.t. } \boldsymbol{x} \geq 0. \tag{2.18}$$

We name Equation (2.18) the l_1*NNLS* model. It is more flexible than NNLS, because it can produce more sparse coefficients as controlled by λ. This model in fact uses the following prior:

$$p(\boldsymbol{x}) = \begin{cases} \frac{1}{\gamma^k} e^{-\frac{\|\boldsymbol{x}\|_1}{\gamma}} & \text{if } \boldsymbol{x} \geq 0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

Now, we give the generalized sparse-coding-based classification approach in details. The method is depicted in Algorithm 2.1. We shall later discuss the optimization algorithms, required in the first step. The NN rule mentioned in Algorithm 2.1 is inspired by the usual way of using NMF as a clustering method. Suppose there are C classes with labels $1, \cdots, C$. For a given new instance $\boldsymbol{b}$, its class is $l = \arg\max_{i=1,\cdots,k} x_i$ (where $k = n$ is the number of training samples). It selects the maximum coefficient in the coefficient vector, say x_l, and then assigns the class label of the corresponding training instance, say c_l, to this new instance. Essentially, this rule is equivalent to applying *nearest neighbor* (NN) classifier in the column space of the training instances. In this space, the representations of the training instances are identity matrix. The NN rule can be further generalized to the weighted K-*NN* rule. Suppose a K-length vector $\bar{\boldsymbol{x}}$ accommodates the K-largest coefficients from

$\boldsymbol{x}$, and $\bar{\boldsymbol{c}}$ has the corresponding K class labels. The class label of $\boldsymbol{b}$ can be designated as $l = \arg\max_{i=1,\cdots,C} s_i$ where $s_i = \text{sum}(\delta_i(\bar{\boldsymbol{x}}))$. $\delta_i(\bar{\boldsymbol{x}})$ is a K-length vector and is defined as

$$(\delta_i(\bar{\boldsymbol{x}}))_j = \begin{cases} \bar{x}_j & \text{if } \bar{c}_j = i, \\ 0 & \text{otherwise .} \end{cases} \tag{2.20}$$

The maximum value of K can be k, the number of dictionary atoms. In this case, K is in fact the number of all non-zeros in $\boldsymbol{x}$. Alternatively, the *nearest subspace* (NS) rule, proposed in [21], can be used to interpret the sparse coding. NS rule has the advantage of the discrimination property in the sparse coefficients. It assigns the class with the minimum regression residual to $\boldsymbol{b}$. Mathematically, it is expressed as $j = \min_{1\le i\le C} r_i(\boldsymbol{b})$ where $r_i(\boldsymbol{b})$ is the regression residual corresponding to the i-th class and is computed as $r_i(\boldsymbol{b}) = \|\boldsymbol{b} - \boldsymbol{A}\delta_i(\boldsymbol{x})\|_2^2$, where $\boldsymbol{\delta}_i(\boldsymbol{x})$ is defined analogously as in Equation (2.20).

Algorithm 2.1 *Sparse-Coding Based Classification*

Input: $\boldsymbol{A}_{m\times n}$: n training instances, $\boldsymbol{c}$: class labels, $\boldsymbol{B}_{m\times p}$: p new instances
Output: $\boldsymbol{p}$: predicted class labels of the p new instances

1. Normalize each instance to have unit l_2-norm;
2. Learn the sparse coefficient matrix $\boldsymbol{X}$, of the new instances by solving Equation (2.9), (2.16), or (2.18);
3. Use a sparse interpreter to predict the class labels of new instances, e.g. the NN, K-NN, or NS rule;

4 Optimization for Sparse Coding [4]

In this section, we focus on the optimizations for the models of sparse coding. We first show that the optimizations are essentially quadratic programmes and dimension-free. We then review the existing methods including interior-point method and proximal method. After that, we propose our active-set methods and decomposition methods. Finally, we state that the all these methods can be extended to their kernel versions.

[4] This section is based on our publications [3] and [22].

4.1 The Optimizations are Quadratic Programmes and Dimension-Free

The problem in Equation (2.9) is equivalent to the following unconstrained non-smooth (that is undifferentiable at some points) *quadratic programming* (QP):

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \lambda\|\boldsymbol{x}\|_1, \tag{2.21}$$

where $\boldsymbol{H}_{k\times k} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, and $\boldsymbol{g} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$. We thus know that the l_1LS problem is a l_1QP problem. This can be converted to the following smooth (that is differentiable everywhere) constrained QP problem:

$$\min_{\boldsymbol{x},\boldsymbol{u}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u} \text{ s.t. } -\boldsymbol{u} \le \boldsymbol{x} \le \boldsymbol{u}, \tag{2.22}$$

where $\boldsymbol{u}$ is an auxiliary vector variable to squeeze $\boldsymbol{x}$ towards zero. It can be further written into the standard form:

$$\begin{aligned} &\min_{\boldsymbol{x},\boldsymbol{u}} \frac{1}{2}[\boldsymbol{x}^{\mathrm{T}}, \boldsymbol{u}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{H} & \boldsymbol{0}_{k\times k} \\ \boldsymbol{0}_{k\times k} & \boldsymbol{0}_{k\times k} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u} \\ &s.t. \begin{bmatrix} \boldsymbol{I}_{k\times k} & -\boldsymbol{I}_{k\times k} \\ -\boldsymbol{I}_{k\times k} & -\boldsymbol{I}_{k\times k} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{u} \end{bmatrix} \le 0, \end{aligned} \tag{2.23}$$

where $\boldsymbol{0}_{k\times k}$ is a null square matrix, $\boldsymbol{I}_{k\times k}$ is an identity matrix. Obviously, the Hessian in this problem is positive semidefinite as we always suppose $\boldsymbol{H}$ is positive semidefinite in order to guarantee that the optimization is convex in this chapter. A symmetric squares matrix $\boldsymbol{H}_{k\times k}$ is said to be positive semidefinite, if and only if $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}_{k\times k}\boldsymbol{x} \ge 0$, $\forall \boldsymbol{x} \in \mathbb{R}^k$.

Both the NNLS problem in Equation (2.16) and the l_1NNLS problem in Equation (2.18) can be easily reformulated to the following *non-negative QP* (NNQP) problem:

$$\min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} \text{ s.t. } \boldsymbol{x} \ge 0, \tag{2.24}$$

where $\boldsymbol{H} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{g} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b}$ for NNLS, and $\boldsymbol{g} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{b} + \boldsymbol{\lambda}$ for l_1NNLS.

4.2 Interior-Point Method

The log-barrier interior-point method for the l_1LS problem (Equation (2.9)) is proposed in [23]. The basic idea is to convert non-smooth problem into unconstrained problem. For the convenience of discussion, we extend this method to solve the l_1QP and NNQP problems. The interior-point method for l_1QP is introduced in the following. Due to similarity of

derivation, we omit the interior-point method for NNQP problem. The problem in Equation (2.22) can be transformed into minimizing the unconstrained log-barrier function:

$$f_t(\boldsymbol{x},\boldsymbol{u}) = t(\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}) - \sum_{i=1}^{k}\log(u_i^2 - x_i^2), \tag{2.25}$$

where $-\sum_{i=1}^{k}\log(u_i^2 - x_i^2)$ is the barrier penalty function, and t is positive parameter to control the penalty.

We can obtain its gradient as below

$$\boldsymbol{d} = \begin{bmatrix} (\frac{\partial f_t(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}})^T \\ (\frac{\partial f_t(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}})^T \end{bmatrix} = \begin{bmatrix} t(\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}) + 2[\frac{x_1}{u_1^2 - x_1^2} \cdots \frac{x_k}{u_k^2 - x_k^2}]^{\mathrm{T}} \\ t\boldsymbol{\lambda} - 2[\frac{u_1}{u_1^2 - x_1^2} \cdots \frac{u_k}{u_k^2 - x_k^2}]^{\mathrm{T}} \end{bmatrix}. \tag{2.26}$$

And its Hessian matrix is

$$\boldsymbol{H} = \begin{bmatrix} \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}^2} & \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}\partial \boldsymbol{x}} \\ \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}\partial \boldsymbol{u}} & \frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}^2} \end{bmatrix}, \tag{2.27}$$

Where

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}^2} = t\boldsymbol{H} + 2\mathrm{diag}(\frac{u_1^2 + x_1^2}{(u_1^2 - x_1^2)^2} \cdots \frac{u_k^2 + x_k^2}{(u_k^2 - x_k^2)^2}), \tag{2.28}$$

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{u}^2} = 2\mathrm{diag}(\frac{u_1^2 + x_1^2}{(u_1^2 - x_1^2)^2} \cdots \frac{u_k^2 + x_k^2}{(u_k^2 - x_k^2)^2}), \tag{2.29}$$

and

$$\frac{\partial f_t^2(\boldsymbol{x},\boldsymbol{u})}{\partial \boldsymbol{x}\partial \boldsymbol{u}} = -4\mathrm{diag}(\frac{x_1 u_1}{(u_1^2 - x_1^2)^2} \cdots \frac{x_k u_k}{(u_k^2 - x_k^2)^2}). \tag{2.30}$$

The optimization of the log-barrier method is to solve a sequence of $f_t(\boldsymbol{x},\boldsymbol{u})$ (Equation (2.25)), as t increases. Newton's method can be used to minimize the unconstrained problem in Equation (2.25), given a positive t [24]. As t increases to positive infinite, the solution converges to the optimal solution. Given a value of t, the convergence rate of Newton's method is super-linear or quadratic with heavy steps.

4.3 Proximal Method

As one of fast first-order methods, the proximal method was proposed in [25] to solve the l_1LS problem. Compared with the interior-point method, it has two advantages. First, the Hessian is not recomputed in each step. Second, an analytical solution can be obtained in each step. As in the interior-point method, we extend the proximal method for the l_1QP problem for the convenience of discussion. The main idea is in the following. The proximal

method works in an iterative procedure. Given the current estimate $\hat{\boldsymbol{x}}$, the new estimate is the solution of the following first-order Taylor series of Equation (2.21):

$$\min_{\boldsymbol{x}} f(\hat{\boldsymbol{x}}) + \nabla q(\hat{\boldsymbol{x}})(\boldsymbol{x} - \hat{\boldsymbol{x}}) + \lambda\|\boldsymbol{x}\|_1 + \frac{L}{2}\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|_2^2, \tag{2.31}$$

where $q(\hat{\boldsymbol{x}}) = \frac{1}{2}\hat{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{H}\hat{\boldsymbol{x}} + \boldsymbol{g}^{\mathrm{T}}\hat{\boldsymbol{x}}$ and $L > 0$ is a parameter. This is done iteratively until termination criteria are met. Equation (2.31) is equivalent to the following proximal operator:

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_2^2 + \eta\|\boldsymbol{x}\|_1, \tag{2.32}$$

where $\bar{\boldsymbol{x}} = \hat{\boldsymbol{x}} - \frac{1}{L}\nabla q(\hat{\boldsymbol{x}})$, and $\eta = \frac{\lambda}{L}$. Unlike the interior-point method with *heavy* Newton step in each iteration where the Hessian is involved in the update, it has element-wise analytical solution:

$$x_i = \begin{cases} \mathrm{sign}(\bar{x}_i)(|\bar{x}_i| - \eta) & \text{if } |\bar{x}_i| > \eta \\ 0 & \text{otherwise} \end{cases}. \tag{2.33}$$

The proximal method for l_1QP has *linear* convergence rate with *swift* steps where gradient is only involved.

4.4 Active-Set Algorithm for l_1LS

A general active-set algorithm for convex QP is provided in [26]. However, the general algorithm may be inefficient for solving the l_1QP problem because its constraints are sparse. Below we introduce the original active-set algorithm [26], and then revise it to solve the l_1QP problem specifically.

Original Active-Set Algorithm for Convex QP

The original active-set method for convex QP is described in Algorithm 2.2 in our own language. A convex QP is expressed as

$$\begin{aligned} \min_{\boldsymbol{x}} & \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} \\ \text{s.t. } & \boldsymbol{A}_{\mathcal{E}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{E}}, \\ & \boldsymbol{A}_{\mathcal{I}}\boldsymbol{x} \leq \boldsymbol{b}_{\mathcal{I}}, \end{aligned} \tag{2.34}$$

where $\boldsymbol{H}$ is semidefinite, $\mathcal{E}$ and $\mathcal{I}$ are the sets of indices of equality and inequality constraints, respectively. The corresponding Lagrange function can be written as

$$L(\boldsymbol{x}, \boldsymbol{\mu}_{\mathcal{E}}, \boldsymbol{\mu}_{\mathcal{I}}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\mu}_{\mathcal{E}}^{\mathrm{T}}(\boldsymbol{A}_{\mathcal{E}}\boldsymbol{x} - \boldsymbol{b}_{\mathcal{E}}) + \boldsymbol{\mu}_{\mathcal{I}}^{\mathrm{T}}(\boldsymbol{A}_{\mathcal{I}}\boldsymbol{x} - \boldsymbol{b}_{\mathcal{I}}), \tag{2.35}$$

where $\boldsymbol{\mu}_{\mathcal{E}}$ and $\boldsymbol{\mu}_{\mathcal{I}}$ are Lagrange multipliers. The *Karush-Kuhn-Tucker* (KKT) conditions of this problem are

$$\begin{cases} \boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} + \boldsymbol{A}_{\mathcal{E}}^{\mathrm{T}}\boldsymbol{\mu}_{\mathcal{E}} + \boldsymbol{A}_{\mathcal{I}}^{\mathrm{T}}\boldsymbol{\mu}_{\mathcal{I}} = 0 \\ \boldsymbol{\mu}_{\mathcal{I}} * (\boldsymbol{A}_{\mathcal{I}}\boldsymbol{x} - \boldsymbol{b}_{\mathcal{I}}) = 0 \\ \boldsymbol{\mu}_{\mathcal{I}} \geq 0 \\ \boldsymbol{A}_{\mathcal{E}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{E}} \\ \boldsymbol{A}_{\mathcal{I}}\boldsymbol{x} \leq \boldsymbol{b}_{\mathcal{I}} \end{cases} . \tag{2.36}$$

The main idea of the original active-set method is that, a working set is updated iteratively until it meets the true active set. In the following, we explain how the algorithm is designed. Suppose the current feasible solution is $\boldsymbol{x}'$ and the working set is $\mathcal{R}$. In order to test the optimality of $\boldsymbol{x}'$ for the objective of Equation (2.34) constrained by the constraints $\boldsymbol{A}_{\mathcal{R}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{R}}$, we need to test if $\boldsymbol{x}'$ can be updated further by adding $\boldsymbol{p}$. If the update $\boldsymbol{p}$ is zero, and the KKT conditions in Equation (2.36) is satisfied, then $\boldsymbol{x}'$ is an optimal solution, and the algorithm terminates successfully. Now, we show how to compute $\boldsymbol{p}$ and how to check the KKT conditions. Suppose the next solution is $\boldsymbol{x} = \boldsymbol{x}' + \boldsymbol{p}$. Substituting it into the objective of Equation (2.34), we have

$$\begin{aligned} &\min_{\boldsymbol{p}} \frac{1}{2}\boldsymbol{p}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{p} + (\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g})^{\mathrm{T}}\boldsymbol{p} \\ &\text{s.t. } \boldsymbol{A}_{\mathcal{R}}\boldsymbol{p} = \boldsymbol{b}_{\mathcal{R}}. \end{aligned} \tag{2.37}$$

If $\boldsymbol{p}$ is zero, then we check if the KKT conditions in Equation (2.36) are satisfied. We can set the Lagrange multipliers corresponding to $\mathcal{P}$ to zero, then the second line of Equation (2.36) corresponding to $\mathcal{P}$ are satisfied. The fourth and fifth lines of Equation (2.36) are certainly satisfied, as $\boldsymbol{x}'$ is feasible. Further, we need to check the Lagrange multipliers corresponding to $\mathcal{R} - \mathcal{E}$. If all of them are non-negative, then we can say that $\boldsymbol{x}'$ is an optimal solution to Equation (2.34). If some of the multipliers corresponding to $\mathcal{R} - \mathcal{E}$ are negative, then $\boldsymbol{x}'$ is not an optimal solution, thus the index corresponding to the worst multiplier should be moved to $\mathcal{P}$. These multipliers corresponding to $\mathcal{R} - \mathcal{E}$ can be obtained by solving the

optimality condition of $\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}$, s.t. $\boldsymbol{A}_{\mathcal{R}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{R}}$. This optimality condition is

$$\boldsymbol{A}_{\mathcal{R}}^{\mathrm{T}}\hat{\boldsymbol{\mu}} = -\boldsymbol{H}\boldsymbol{x} - \boldsymbol{g}, \tag{2.38}$$

where $\hat{\boldsymbol{\mu}} = \boldsymbol{\mu}_{\mathcal{R}}$ is the current multipliers corresponding to the feasible solution $\mathcal{R}$. Note constraint $\boldsymbol{A}_{\mathcal{R}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{R}}$ is certainly satisfied, as $\boldsymbol{p} = 0$.

If $\boldsymbol{p}$ is not zero, $\boldsymbol{x}' + \alpha\boldsymbol{p}$, for all positive step size α, is definitely feasible for the constrains indexed by $\mathcal{R}$, but may violate the inequality constrains indexed by $\mathcal{P}$. We need to find a α such that all inequality constraints hold. From the desired inequality constraints $\boldsymbol{a}_i^{\mathrm{T}}(\boldsymbol{x}' + \alpha\boldsymbol{p}) \leq b_i$ ($i \in \mathcal{P}$ and $\boldsymbol{a}_i^{\mathrm{T}}$ is a row of $\boldsymbol{A}_{\mathcal{P}}$), we can see that, if $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p} \leq 0$, then any $\alpha \in [0, 1]$ can be used to satisfy the inequality constraints (we limit the maximum step size to 1 in the algorithm). If $\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p} > 0$, then some inequality may be violated. In this case, the maximum step size satisfying all inequality constraints is

$$\alpha = \min\big(1, \beta\big), \tag{2.39}$$

where $\beta = \min_{i \in \mathcal{P}, \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p} \geq 0} \frac{b_i - \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}$. If $\beta \leq 1$, the inequality constraints corresponding to β become active, and are called blocking constraints. The index of a blocking constraint should be added to the working set $\mathcal{R}$.

Revised Active-Set Algorithm for l_1LS

In order to solve our specific problem efficiently in Equation (2.23), we have to modify the general method, because i) our constraint is sparse, and for the i-th constraint, we have $x_i - u_i \leq 0$ (if $i \leq k$) or $-x_i - u_i \leq 0$ (if $i \geq k + 1$); and ii) when u_i is not constrained in the current working set, the QP constrained by the working set is unbounded, therefore it is not necessary to solve this problem to obtain $\boldsymbol{p}_t$. In the latter situation, $\boldsymbol{p}_t$ is unbounded. This could cause some issues in numerical computation. Solving the unbounded problem is time-consuming if the algorithm is unaware of the unbounded issue. If $\boldsymbol{p}_t$ contains values equal to $\pm\infty$, then the algorithm may crash.

In Algorithm 2.3, we revise the active-set algorithm for l_1LS sparse coding. To address the potential issues above, we have the following four modifications. First, we require that the working set is *complete*. That is, all the variables in $\boldsymbol{u}$ must be constrained when computing the current update step. And therefore all variables in $\boldsymbol{x}$ are also constrained due to the specific structure of the constraints in our problem. For example, if $k = 3$, a working set $\{1, 2, 6\}$ is complete as all variables, $x_1, x_2, x_3, u_1, u_2, u_3$, are constrained, while $\{1, 2, 4\}$ is not complete, as u_3 (and x_3) is not constrained. Second, the update step of the

Algorithm 2.2 Original Active-Set Algorithm for Convex QP [26]

Input: semidefinite Hessian $\boldsymbol{H}_{k\times k}$, vector $\boldsymbol{g}_{k\times 1}$, scalar λ

Output: vector $\boldsymbol{x}$ which is an optimal solution to $\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}$, s.t. $\boldsymbol{A}_{\mathcal{E}}\boldsymbol{x} = \boldsymbol{b}_{\mathcal{E}},\ \boldsymbol{A}_{\mathcal{I}}\boldsymbol{x} \leq \boldsymbol{b}_{\mathcal{I}}$

{initialize the algorithm by a feasible point and a working set}
Initialize with a feasible point $\boldsymbol{x}_0$;
$\mathcal{R} = \{i | \forall i \in \mathcal{E} \cup \mathcal{I} : \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x} = 0\}$; {initialize working set}
$\mathcal{P} = \mathcal{E} \cup \mathcal{I} - \mathcal{R}$; {place the remaining to the inactive(passive) set}

while true **do**
  {compute update step}
  Solve $\min_{\boldsymbol{p}} \frac{1}{2}\boldsymbol{p}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{p} + (\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g})^{\mathrm{T}}\boldsymbol{p}$ s.t. $\boldsymbol{A}_{\mathcal{R}}\boldsymbol{p} = \boldsymbol{b}_{\mathcal{R}}$;

  if $\boldsymbol{p} = 0$ **then**
    Compute Lagrange multiplier that satisfy $\boldsymbol{A}_{\mathcal{R}}^{\mathrm{T}}\hat{\boldsymbol{\mu}} = -\boldsymbol{H}\boldsymbol{x} - \boldsymbol{g}$;
    if $\hat{\mu}_i \geq 0\ \forall i \in \mathcal{R} \cap \mathcal{I}$ **then**
      Return $\boldsymbol{x}$ as an optimal solution;
    else
      $\mathcal{R} = \mathcal{R} - j$; $\mathcal{P} = \mathcal{P} + j$, where $j = \arg\min_{l \in \mathcal{R}\cap\mathcal{I}} \hat{\mu}_l$;
    end if
  end if

  if $\boldsymbol{p} \neq 0$ **then**
    $\alpha = \min\left(1, \beta\right)$ where $\beta = \min_{i\in\mathcal{P}, \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p} \geq 0} \frac{b_i - \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}$; $\{\boldsymbol{a}_i^{\mathrm{T}}$ is a row of $\boldsymbol{A}_{\mathcal{P}}\}$
    $\boldsymbol{x} = \boldsymbol{x} + \alpha\boldsymbol{p}$;
    {if there are blocking constraints}
    if $\beta \leq 1$ **then**
      $\mathcal{R} = \mathcal{R} + j$; $\mathcal{P} = \mathcal{P} - j$, where $j = \arg\min_{i\in\mathcal{P}, \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p} \geq 0} \frac{b_i - \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{x}}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}$;
    end if
  end if
end while

variables that are constrained once in the working set are computed by solving the equality constrained QP. The variables constrained twice are directly set to zeros. In the example above, suppose the current working set is $\{1, 2, 4, 6\}$, then x_2, x_3, u_2, u_3 are computed by the constrained QP, while x_1 and u_1 are zeros. This is because the only value satisfying the constraint $-u_1 = x_1 = u_1$ is $x_1 = u_1 = 0$. Third, in this example, we do not need to solve the equality constrained QP with four variables. In fact we only need two variables by setting $u_2 = -x_2$ and $u_3 = x_3$. Fourth, once a constraint is dropped from the working set and it becomes incomplete, other inequalities must be immediately added to it until it is complete. In the initialization of Algorithm 2.3, we can alternatively initialize $\boldsymbol{x}$ by 0's. This is more efficient than initializing by $\boldsymbol{x} = (\boldsymbol{H})^{-1}(-\boldsymbol{g})$ for large-scale and very sparse problems.

4.5 Active-Set Algorithm for NNLS and l_1NNLS

Now, we present the active-set algorithm for NNQP. This problem is easier to solve than l_1QP as the scale of the Hessian of NNQP is half that of l_1QP and the constraint is much simpler. Our algorithm is obtained through generalizing the famous active-set algorithm for NNLS originally by [27]. The generalized algorithm is given in Algorithm 2.4. The warm-start point is initialized by the solution to the unconstrained QP. As in Algorithm 2.3, $\boldsymbol{x}$ can be alternatively initialized by 0's. The algorithm keeps adding and dropping constraints in the working set until the true active set is found.

4.6 Parallel Active-Set Algorithms

The formulations of l_1QP and NNQP sparse coding for p new instances are, respectively,

$$\begin{aligned} &\min_{\boldsymbol{X},\boldsymbol{U}} \sum_{i=1}^{p} \frac{1}{2}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{x}_i + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}_i, \\ &\text{s.t. } -\boldsymbol{U} \leq \boldsymbol{X} \leq \boldsymbol{U}, \end{aligned} \tag{2.42}$$

and

$$\min_{\boldsymbol{X}} \sum_{i=1}^{p} \frac{1}{2}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{x}_i \text{ s.t. } \boldsymbol{X} \geq 0. \tag{2.43}$$

If we want to classify multiple new instances using the classification method described in Algorithm 2.1, the initial idea in [21] and [20] is to optimize the sparse coding one at a time. The interior-point algorithm, proposed in [23], is a fast large-scale sparse coding

Algorithm 2.3 Active-Set Algorithm for l_1QP

Input: Hessian $\boldsymbol{H}_{k\times k}$, vector $\boldsymbol{g}_{k\times 1}$, scalar λ

Output: vector $\boldsymbol{x}$ which is a solution to $\min_{\boldsymbol{x},\boldsymbol{u}} \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} + \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{u}$, s.t. $-\boldsymbol{u} \le \boldsymbol{x} \le \boldsymbol{u}$

{initialize the algorithm by a feasible solution and a complete working set}
$\boldsymbol{x} = (\boldsymbol{H})^{-1}(-\boldsymbol{g})$; $\boldsymbol{u} = |\boldsymbol{x}|$;
$\mathcal{R} = \{i, j|\forall i:$ if $x_i > 0$ let $j = k + i$ otherwise $j = i\}$; {initialize working set}
$\mathcal{P} = \{1 : 2k\} - \mathcal{R}$; {initialize inactive(passive) set}

while true **do**
{compute update step}
Let $\mathcal{R}_{\text{once}}$ be the indices of variables constrained once by $\mathcal{R}$;
$\boldsymbol{p}_{2k\times 1} = 0$;

$$\boldsymbol{p}_{\boldsymbol{x},\mathcal{R}_{\text{once}}} = \arg\min_{\boldsymbol{q}} \boldsymbol{q}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{R}_{\text{once}}}\boldsymbol{q} + [\boldsymbol{H}_{\mathcal{R}_{\text{once}}}\boldsymbol{x}_{\mathcal{R}_{\text{once}}} + \boldsymbol{g}_{\mathcal{R}_{\text{once}}} + \lambda\boldsymbol{e}]^{\mathrm{T}}\boldsymbol{q}, \tag{2.40}$$

where $e_i = 1$ if $u_{\mathcal{R}_{\text{once}},i} = x_{\mathcal{R}_{\text{once}},i}$, or -1 if $u_{\mathcal{R}_{\text{once}},i} = -x_{\mathcal{R}_{\text{once}},i}$;

if $\boldsymbol{p} = 0$ **then**
Obtain Lagrange multiplier $\hat{\boldsymbol{\mu}}$ by solving

$$\boldsymbol{A}_{\mathcal{R}}^{\mathrm{T}}\hat{\boldsymbol{\mu}} = -\begin{bmatrix}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} \\ \boldsymbol{\lambda}\end{bmatrix}, \tag{2.41}$$

where $\boldsymbol{A}$ is the constraint matrix in Equation (2.23);
if $\hat{\mu}_i \ge 0\ \forall i \in \mathcal{R}$ **then**
Terminate successfully;
else
$\mathcal{R} = \mathcal{R} - j$; $\mathcal{P} = \mathcal{P} + j$ where $j = \arg\min_{l\in\mathcal{R}} \mu_l$;
Add other passive constraints to $\mathcal{R}$ until it is complete;
end if
end if

if $\boldsymbol{p} \ne 0$ **then**
$\alpha = \min\big(1, \beta\big)$, where $\beta = \min_{i\in\mathcal{P}, \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}\ge 0} \frac{-\boldsymbol{a}_i^{\mathrm{T}}[\boldsymbol{x};\boldsymbol{u}]}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}$; {$\boldsymbol{a}_i^{\mathrm{T}}$ is a row of $\boldsymbol{A}_{\mathcal{P}}$}
$[\boldsymbol{x};\boldsymbol{u}] = [\boldsymbol{x};\boldsymbol{u}] + \alpha\boldsymbol{p}$;
{if there are blocking constraints}
if $\beta \le 1$ **then**
$\mathcal{R} = \mathcal{R} + j$; $\mathcal{P} = \mathcal{P} - j$, where $j = \arg\min_{i\in\mathcal{P}, \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}\ge 0} \frac{-\boldsymbol{a}_i^{\mathrm{T}}[\boldsymbol{x};\boldsymbol{u}]}{\boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{p}}$;
end if
end if
end while

Algorithm 2.4 Active-Set NNQP Algorithm

Input: Hessian $\boldsymbol{H}_{k\times k}$, vector $\boldsymbol{g}_{k\times 1}$

Output: vector $\boldsymbol{x}$ which is a solution to $\min \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}^{\mathrm{T}}\boldsymbol{x}$, s.t. $\boldsymbol{x}\geq 0$

$\boldsymbol{x}=[(\boldsymbol{H})^{-1}(-\boldsymbol{g})]_+$; { $\boldsymbol{x}=[\boldsymbol{y}]_+$ is defined as $x_i=y_i$ if $y_i>0$, otherwise $x_i=0$}
$\mathcal{R}=\{i|x_i=0\}$; {initialize active set}
$\mathcal{P}=\{i|x_i>0\}$; {initialize inactive(passive) set}
$\boldsymbol{\mu}=\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}$; {the lagrange multiplier}

while $\boldsymbol{R}\neq\emptyset$ and $\min_{i\in\mathcal{R}}(\mu_i)<-\epsilon$ **do**
  {ϵ is a small positive numerical tolerance}
  $j=\arg\min_{i\in\mathcal{R}}(\mu_i)$; {get the minimal negative multiplier}
  $\mathcal{P}=\mathcal{P}+\{j\}$; $\mathcal{R}=\mathcal{R}-\{j\}$;
  $\boldsymbol{t}_{\mathcal{P}}=(\boldsymbol{H}_{\mathcal{P}})^{-1}(-\boldsymbol{g}_{\mathcal{P}})$;
  $\boldsymbol{t}_{\mathcal{R}}=\boldsymbol{0}$;

  while $\min \boldsymbol{t}_{\mathcal{P}}\leq 0$ **do**
    $\alpha=\min_{i\in\mathcal{P},t_i\leq 0}\frac{x_i}{x_i-t_i}$;
    $\mathcal{K}=\arg\min_{i\in\mathcal{P},t_i\leq 0}\frac{x_i}{x_i-t_i}$; {there is one or several indices correspond to α}
    $\boldsymbol{x}=\boldsymbol{x}+\alpha(\boldsymbol{t}-\boldsymbol{x})$;
    $\mathcal{P}=\mathcal{P}-\mathcal{K}$; $\mathcal{R}=\mathcal{R}+\mathcal{K}$;
    $\boldsymbol{t}_{\mathcal{P}}=(\boldsymbol{H}_{\mathcal{P}})^{-1}(-\boldsymbol{g}_{\mathcal{P}})$;
    $\boldsymbol{t}_{\mathcal{R}}=\boldsymbol{0}$;
  end while

  $\boldsymbol{x}=\boldsymbol{t}$;
  $\boldsymbol{\mu}=\boldsymbol{H}\boldsymbol{x}+\boldsymbol{g}$;
end while

algorithm, and the proximal algorithm in [25] is a fast first-order method whose advantages have been recently highlighted for non-smooth problems. If we adapt both algorithms to solve our multiple l_1QP in Equation (2.42) and NNQP in Equation (2.43), it will be difficult to solve the single problems in parallel and share the computations. Therefore, the time-complexity of the multiple problems will be the summation of that of the individual problems. However, the multiple problems can be much more efficiently solved by active-set algorithms. We adapt both Algorithms 2.3 and 2.4 to solve multiple l_1QP and NNQP in a parallel fashion. The individual active-set algorithms can be solved in parallel by sharing the computation of matrix inverses (systems of linear equations in essence). At each iteration, single problems having the same active set have the same systems of linear equations to solve. These systems of linear equations can be solved once only. For a large value p, that is large-scale multiple problems, the active-set algorithms have dramatic computational advantage over interior-point [23] and proximal [25] methods unless these methods have a scheme for sharing computations. Additionally, active-set methods are more precise than interior-point methods. Interior-point methods do not allow $u_i^2 = x_i^2$ and u_i^2 must be always greater than x_i^2 due to feasibility. But $u_i^2 = x_i^2$ is naturally possible when the i-th constraint is active. $u_i = x_i = 0$ is reasonable and possible. The active-set algorithms do allow this situation.

4.7 Decomposition Method [5]

A decomposition method [28] has first been devised in the optimization of large-scale SVM which is a QP problem constrained by equality and bound constraints. The basic idea of the decomposition method is, in fact, an implementation of the block-coordinate-descent scheme [29]. The decomposition method works in an iterative procedure. In each iteration, a few number of variables violating the optimality conditions (e.g. KKT conditions) are included in a working set, while the rest are fixed. Only the variables in the working set are updated by a solver. This procedure iterates until no coefficient violates the KKT conditions. Because the objective is decreased in each iteration, the convergence to the optimal solution is guaranteed in regular case. *Sequential minimal optimization* (SMO) [30] is the extreme case of the decomposition method for SVM, where only a minimal number of variables (two variables) are updated. One of the features of the SMO method for SVM is that the subproblem with only two variables can be solved analytically. See Section 4 of Appendix B for more details of decomposition methods for SVMs.

[5] This section is our collaborative research [22] with Dr. Richard J. Caron, Department of Mathematics and Statistics, University of Windsor.

In this section, we propose decomposition methods for the l_1LS and NNLS sparse coding models. Our contributions are as follows

1. To the best of our knowledge, it is the first time that the idea of decomposition is investigated in sparse coding. The decomposition method has been successfully applied in the optimization of support vector machine (SVM).

2. We design sequential minimal optimization (SMO) methods for l_1LS, NNLS, and l_1NNLS sparse coding models, but our methods may be applicable to many other models.

4.8 Decomposition Method for l_1QP

Let $\mathcal{A}$ be the set of a few working variables and $\mathcal{P}$ be the set of fixed variables. The decomposition of $f(\boldsymbol{x})$ in Equation (2.21) can be

$$\begin{aligned} f(\boldsymbol{x}_{\mathcal{A}}) &= \frac{1}{2}[\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}, \boldsymbol{x}_{\mathcal{P}}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{H}_{\mathcal{A}\mathcal{A}} & \boldsymbol{H}_{\mathcal{A}\mathcal{P}} \\ \boldsymbol{H}_{\mathcal{P}\mathcal{A}} & \boldsymbol{H}_{\mathcal{P}\mathcal{P}} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} + [\boldsymbol{g}_{\mathcal{A}}^{\mathrm{T}}, \boldsymbol{g}_{\mathcal{P}}^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} + \lambda \| \begin{bmatrix} \boldsymbol{x}_{\mathcal{A}} \\ \boldsymbol{x}_{\mathcal{P}} \end{bmatrix} \|_1 \\ &= \frac{1}{2}\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{A}\mathcal{A}}\boldsymbol{x}_{\mathcal{A}} + (\boldsymbol{H}_{\mathcal{A}\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + \boldsymbol{g}_{\mathcal{A}})^{\mathrm{T}}\boldsymbol{x}_{\mathcal{A}} + \lambda\|\boldsymbol{x}_{\mathcal{A}}\|_1 + constant. \end{aligned} \tag{2.44}$$

We can see that the subproblem corresponding to the working set is a l_1QP problem as well. From this point, a solver of l_1QP is still required. In this study, we focus on the extreme case of Equation (2.44): the SMO method. It can be easily seen that the minimal size of $\mathcal{A}$ is *one* and $\mathcal{P}$ contains the remaining $k-1$ variables. Without loss of generality, we denote such variable by x_1. This results in the following subproblem:

$$\min_{x_1} f(x_1) = \frac{1}{2}h_{11}x_1^2 + b_1x_1 + \lambda|x_1|, \tag{2.45}$$

where $b_1 = \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_1$.

Analytical Solution

The SMO problem in Equation (2.45) can be solved analytically. Now let us separate the interval into $x_1 \geq 0$ and $x_1 \leq 0$. For $x_1 \geq 0$, the objective $f(x_1)$ becomes

$$f(x_1) = \frac{1}{2}h_{11}x_1^2 + (b_1 + \lambda)x_1. \tag{2.46}$$

Taking first-order derivative and setting it to zero, we have

$$\frac{\partial f(x_1)}{\partial x_1} = h_{11}x_1 + (b_1 + \lambda) = 0. \tag{2.47}$$

We thus have $x_1^{(+)} = \frac{-b_1-\lambda}{h_{11}}$. Therefore, for interval $x_1 \geq 0$, we have the optimal solution:

$$x_1^{(+*)} = \begin{cases} x_1^{(+)} & \text{if } x_1^{(+)} \geq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.48}$$

Similarly, for $x_1 \leq 0$, the objective $f(x_1)$ becomes

$$f(x_1) = \frac{1}{2}h_{11}x_1^2 + (b_1 - \lambda)x_1. \tag{2.49}$$

We thus have $x_1^{(-)} = \frac{-b_1+\lambda}{h_{11}}$. And for interval $x_1 \leq 0$, we have the optimal solution:

$$x_1^{(-*)} = \begin{cases} x_1^{(-)} & \text{if } x_1^{(-)} \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.50}$$

By considering both positive and negative interval together, the optimal solution to Equation (2.45) is the one, among $x_1^{(+*)}$ and $x_1^{(-*)}$, which obtains the minimum objective value, that is $x_1^* = \arg\min_{\{x_1^{(+*)}, x_1^{(-*)}\}} f(x_1)$.

We note that $x_1^{(+)} \geq 0$ is equivalent to $b_1 \leq -\lambda$, and $x_1^{(-)} \leq 0$ is equivalent to $b_1 \geq \lambda$. We can state that if $b_1 \leq -\lambda$ or $b_1 \geq \lambda$ the solution to Equation (2.45) is $x_1^* = \frac{-b_1-\lambda}{h_{11}}$ or $x_1^* = \frac{-b_1+\lambda}{h_{11}}$, respectively. Otherwise, $x_1^* = 0$. Therefore, the solution to Equation (2.45) can be equivalently written as

$$x_1^* = \begin{cases} \frac{-\text{sign}(b_1)(|b_1|-\lambda)}{h_{11}} & \text{if } |b_1| \geq \lambda \\ 0 & \text{otherwise} \end{cases}. \tag{2.51}$$

From this, we can obtain a general proposition below which is very useful:

Proposition 1. *The solution to the following problem*

$$\min_x f(x) = x^2 + bx + \lambda|x| \tag{2.52}$$

is analytically

$$x^* = \begin{cases} -sign(b)(|b| - \lambda) & if\ |b| \geq \lambda \\ 0 & otherwise \end{cases}. \tag{2.53}$$

Now that we have known how to update a working variable. In the following, we show how to select a working variable.

Select x_1 Which Violates the Optimality Condition

The optimality condition of Equation (2.21) is

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H} + \boldsymbol{g}^{\mathrm{T}} + \nabla\lambda\|\boldsymbol{x}\|_1 = 0. \tag{2.54}$$

However, because $\lambda\|\boldsymbol{x}\|_1$ is not differentiable, we need to resort to the concept of subgradient. We hence have

$$s_i = \boldsymbol{H}_{i:}\boldsymbol{x} + g_i = \begin{cases} \lambda & x_i < 0 \\ -\lambda & x_i > 0 \\ \in [-\lambda, \lambda] & x_i = 0 \end{cases}, \tag{2.55}$$

where $\boldsymbol{H}_{i:}$ is the i-th row of $\boldsymbol{H}$, similarly $\boldsymbol{H}_{:i}$ is the i-th column of $\boldsymbol{H}$. The algorithm proceeds as follows. We iteratively select a variable x_1 which violates the optimality condition in Equation (2.55). In an iteration, x_1 is updated analytically as in Equation (2.51). If all variables satisfy the optimality condition, the algorithm terminates. In order to maximize the effort of violating variable selection, the one which violates the optimality condition the most should be preferred. The extent of violation is measured by the difference, as given below, between s_i and its desired values:

$$d_i = \begin{cases} |s_i - \lambda| & x_i < 0 \\ |s_i + \lambda| & x_i > 0 \\ |s_i| - \lambda & x_i = 0 \end{cases}. \tag{2.56}$$

Update $\boldsymbol{s}$ and Compute b_1

In each iteration after updating x_1, the vector $\boldsymbol{s}$ needs to be updated in order to obtain the optimality condition and select the new violating variable x_1. Intuitively, $\boldsymbol{s}$ can be

updated by its definition in Equation (2.55). However, it would take linear time to update each element s_i. In fact, if we keep a record of its previous value (denoted by s'_i), s_i can be updated in constant time. We denote by $\boldsymbol{x}'$ the coefficients before updating x_1, and by $\boldsymbol{x}$ the coefficients after updating x_1. We know that $s'_i = h_{i1}x'_1 + \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} + g_i$, $s_i = h_{i1}x_1 + \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_i$, and $\boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} = \boldsymbol{H}_{i\mathcal{P}}\boldsymbol{x}_{\mathcal{P}}$. We thus can update s_i by the following equation which takes constant time:

$$s_i = h_{i1}(x_1 - x'_1) + s'_i. \tag{2.57}$$

Similar idea also applies to the computation of b_1 before updating x_1. According to the definition in Equation (2.45), b_1 can be updated in linear time. However, it can actually be updated in constant time as well. We know that $s'_1 = h_{11}x'_1 + \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}'_{\mathcal{P}} + g_1 = h_{11}x' + b_1$. We thus can update b_1 in constant time:

$$b_1 = s'_1 - h_{11}x'. \tag{2.58}$$

Initialize $\boldsymbol{x}$ and $\boldsymbol{s}$

Before the iterative update of the method, $\boldsymbol{x}$ and $\boldsymbol{s}$ need to be initialized. We can initialize $\boldsymbol{x}$ and $\boldsymbol{s}$ by zeros and $\boldsymbol{g}$, respectively. This initialization makes the following iterative update very efficient. This is because $\boldsymbol{x}$ is eventually sparse, which means that $\boldsymbol{x} = \boldsymbol{0}$ is a very good approximate. This initialization strategy may also apply to many other sparse coding methods, for example active-set methods.

4.9 Decomposition Method for NNQP

Now we concisely derive the decomposition method for NNQP (Equation (3.9)). The decomposed objective of NNQP can be formulated into

$$f(\boldsymbol{x}_{\mathcal{A}}) = \frac{1}{2}\boldsymbol{x}_{\mathcal{A}}^{\mathrm{T}}\boldsymbol{H}_{\mathcal{A}\mathcal{A}}\boldsymbol{x}_{\mathcal{A}} + (\boldsymbol{H}_{\mathcal{A}\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + \boldsymbol{g}_{\mathcal{A}})^{\mathrm{T}}\boldsymbol{x}_{\mathcal{A}} + constant. \tag{2.59}$$

Since this objective is constrained only by nonnegativity, its SMO case also has only *one* variable in $\mathcal{A}$. Without loss of generality, we denote such variable as x_1 as above. This results in the following problem:

$$\min_{x_1} f(x_1) = \frac{1}{2}h_{11}x_1^2 + b_1x_1 \text{ s.t. } x_1 \geq 0, \tag{2.60}$$

where $b_1 = \boldsymbol{H}_{1\mathcal{P}}\boldsymbol{x}_{\mathcal{P}} + g_1$. It is easy to obtain the analytical solution to Equation (2.60):

$$x_1^* = \begin{cases} \frac{-b_1}{h_{11}} & \text{if } b_1 \leq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2.61}$$

The Lagrange function of Equation (3.9) is

$$L(\boldsymbol{x}, \boldsymbol{s}) = \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} + \boldsymbol{g}^{\mathrm{T}}\boldsymbol{x} - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{x}, \tag{2.62}$$

where $\boldsymbol{s}$ is the vector of dual variables. Therefore, the corresponding KKT conditions of Equation (3.9) are

$$\begin{cases} \boldsymbol{s} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{g} \\ \boldsymbol{s} * \boldsymbol{x} = 0 \\ \boldsymbol{x} \geq 0 \\ \boldsymbol{s} \geq 0. \end{cases} \tag{2.63}$$

From the KKT conditions, we can find that $s_i = \boldsymbol{H}_{i:}\boldsymbol{x} + g_i$ is the Lagrange multiplier of the i-th primal variable, x_i. For the optimal x_i, the corresponding s_i must fulfil the following conditions:

$$\begin{cases} s_i * x_i = 0 \\ s_i \geq 0 \end{cases}. \tag{2.64}$$

The SMO-based NNQP works in an iterative procedure. Before the iterative loop, $\boldsymbol{x}$ can be initialized by zeros, and $\boldsymbol{s}$ is therefore initialized with $\boldsymbol{g}$. After that, variables violating the KKT conditions in Equation (2.64) are updated in an iterative loop until all variables fulfil the KKT conditions. In each iteration, we need to find one variable which violates the KKT condition in order to update by Equation (2.61); before updating x_1, b_1 can be computed using Equation (2.58); after updating x_1, $\boldsymbol{s}$ can be updated as in Equation (2.57). In our implementation, we select the variable which violates the KKT conditions the most.

The magnitude of violation is measured by the following equation:

$$d_i = \begin{cases} 0 & \text{if } x_i = 0 \text{ and } s_i \geq 0 \\ |s_i| & \text{if } x_i = 0 \text{ and } s_i < 0 \\ |s_i| & \text{if } x_i > 0 \end{cases} . \tag{2.65}$$

4.10 Kernel Extensions

Two main advantages of a kernel method are given as follows. First, it linearize complex patterns in a higher-dimensional feature space. Second, it is dimension-free in optimization and decision making. Dimension-free means that the computation is not explicitly affected by the number of features. The methods require only the inner products between multivariate samples (that is $\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_j$ where $\boldsymbol{d}_i, \boldsymbol{d}_j \in \mathbb{R}^m$). The number of features, m, does not affect the time complexity and spacial complexity at all. In the following, we show that the sparse coding techniques can be extended to kernel versions.

As the optimizations of l_1QP and NNQP only require inner products between the instances instead of the original data, our active-set algorithms can be naturally extended to solve the kernel sparse coding problem by replacing inner products with kernel matrices. The NS decision rule used in Algorithm 2.1 also requires only inner products. And the weighted K-NN rule only needs the sparse coefficient vector and class information. Therefore, the classification approach in Algorithm 2.1 can be extended to kernel version. For narrative convenience, we also denote the classification approaches using l_1LS, NNLS, and l_1NNLS sparse coding as l_1LS, NNLS, and l_1NNLS, respectively. Prefix "K" is used for kernel versions.

5 The Performance of Sparse Coding

5.1 The Performance of Active-Set Sparse Coding for Classification [6]

Two high-throughput biological data, including a microarray gene expression data set and a protein mass spectrometry data set, are used to test the performance of our sparse coding based classification approach. The microarray data set is a collection of gene expression profiles of breast cancer subtypes [31]. This data set includes 158 tumor samples from five subtypes measured on 13582 genes. The mass spectrometry data set is composed of 332

[6] This section is based on our publications [3] and [17].

samples from normal class and prostate cancer class [32]. Each sample has 15154 features, that is the mass-to-charge ratios.

When dictionary learning was not involved, the dictionary was "lazily" composed by all the training available instances. In our experiment, the active-set optimization methods for l_1LS, NNLS, and l_1NNLS were tested. The weighted K-NN rule and NS rule, mentioned in Algorithm 2.1, were compared. We set K in the K-NN rule to the number of all training instances, which is an extreme case as opposite to the NN rule. Linear and *radial basis function* (RBF) kernels were employed. We compared our active-set algorithms with the interior-point [23] method and proximal [25] method for l_1LS sparse coding (abbreviated by l_1LS-IP and l_1LS-PX). Benchmark classifiers, including k-NN and SVM using RBF kernel, were compared. We employed four-fold *cross-validation* [7] to partition a data set into training sets and test sets. All the classifiers ran on the same training and test splits for fair comparison. We performed 20 runs of cross-validation and recorded the averages and standard deviations. Line or grid search was used to select the parameters of a classifiers.

We use accuracy to measure the classification performance. The accuracy is defined as the ratio of the number of correctly predicted test samples to the number of all test samples. The average accuracies of all classifiers with the corresponding standard deviations on both data sets are compared in Figure 2.1, from which we have four observations. First, the weighted K-NN rule obtained comparable accuracies with the NS rule. The advantage of the K-NN rule over the NS rule is that the former predicts the class labels based on the sparse coefficient vector solely, while the latter has to use the training data to compute regression residuals. Therefore, the K-NN rule is more efficient and should be preferred. Second, on the Prostate data, the sparse coding method l_1LS and Kl_1LS achieved the best accuracy. This convinces us that sparse coding based classifiers can be very effective for classifying high-throughput biological data. Third, the non-negative models including NNLS, l_1NNLS and their kernel extensions achieved competitive accuracies with the state-of-the-art SVM on both data set. Fourth, the l_1LS sparse coding using our active-set algorithm had the same accuracy as that using the interior-point algorithm and proximal algorithm on Breast data. But on Prostate data, the proximal method yielded a worse accuracy. This implies that our active-set method converges to the global minima as the interior-point method, while performance may be deteriorated by the approximate solution

[7] The sample size of a data set and time complexity of a method determine the choice of k in k-fold cross-validation. For classification, the proportion of class sizes also affects the value of k. In this thesis, we used $k = 3$ or 4 for relatively large data size and methods of high time-complexity. We used $k = 9$ or 10 for relatively small sample size. The proportion of classes sizes in the training set should approximate to that in the original data set. And a training set should contains samples of all classes.

obtained by the proximal method in practice.

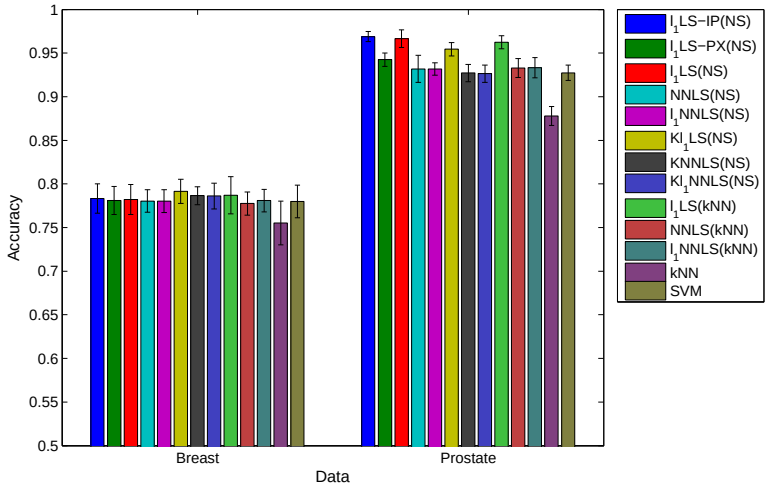


Figure 2.1: Accuracies of sparse coding and benchmark approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation.

The mean running time (in seconds) of cross-validation are shown in Figure 2.2. For better comparison, logarithm of base two was taken on the results. First of all, we can clearly see that the interior-point method is very slow for the l_1LS sparse coding. Second, our active-set method is more efficient than the proximal method on Breast data. This is because i) active-set methods are usually the fastest ones for quadratic and linear programmes of small and medium sizes; and ii) expensive computations, like solving systems of linear equations, can be shared in the active-set method. Third, NNLS and l_1NNLS have the same time-complexity. This is reasonable, because both can be formulated as NNQP problems. These non-negative models are much simpler and faster than the non-smooth l_1LS model. Hence, if similar performance can be obtained by l_1LS and the non-negative models in an application, we should give preference to NNLS and l_1NNLS.

5.2 The Performance of Decomposition Method [8]

We tested our SMO methods on the large-scale microarray data given in [33]. This data set has 5456 samples including healthy tissue samples and cancer samples from 13 different

[8] This section is based on our publication [22].

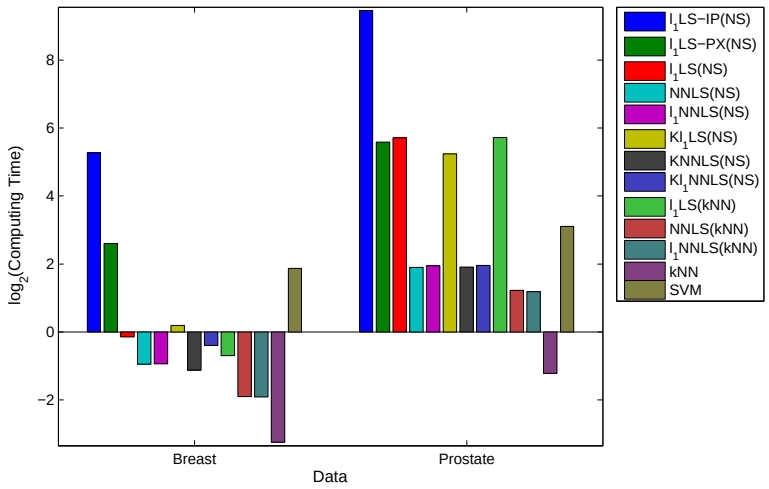


Figure 2.2: Computing time of sparse coding and benchmark methods. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the K-NN rule and NS rule is indicated in the corresponding parentheses for sparse coding classifiers. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest sparse coding models use active-set method without explicit notation.

tissue types. The number of dimensions (genes) of each sample is 9471. We randomly selected 100 samples as the test set, and used the remaining 5356 samples as the training set. In few cases where the computation was very costly, we only used 10 test samples. We put all the training samples in the dictionary $\boldsymbol{A}_{9471\times 5356}$. Therefore, the Hessian is of size 5356 by 5356. We used linear kernel in our experiments. We compared our SMO methods with proximal method and active-set methods. Due to the large numbers of samples and features, the interior-point method proposed in [23] crashed our computer (the implementation in [23] is not dimension-free). We recorded the wall-clock computing time, in seconds, and the number of iterations of the tests, as the value of λ increased. The average results of each method are given in Tables 2.1 and 2.2. The corresponding sparsity is also given in the first table. The sparsity is the percentage of zeros in a coefficient vector. It is defined as $\frac{\text{sum}(\boldsymbol{x}<\varepsilon \max(\boldsymbol{x}))}{k}$, where $\boldsymbol{x}$ is the sparse coefficient vector, k is the length of $\boldsymbol{x}$, and we set $\varepsilon = 0.001$ in our experiment.

From Table 2.1, we have the following observations. First, for all methods the computing time and numbers of iterations decrease gradually as the value of λ increases. Second, for the non-smooth l_1QP model, our SMO method is much faster than the proximal and active-set

methods. Third, for the smooth NNQP model, SMO is also efficient, though the active-set method keeps its efficiency. Fourth, the sparsity increases as the value of λ rises. When $\lambda = 0$, only the non-negativity induces the sparsity in the NNQP model. We can see that very high sparsity can be obtained by using the non-negativity solely.

Table 2.1: Mean computing time (in seconds) of each sample when using 5356 samples as training set.

λ	l_1QP				NNQP		
	SMO	proximal	active-set	sparsity	SMO	active-set	sparsity
0	-	-	-	-	17.99	0.49	0.9954
0.0001	273.78	554.37	-	0.9478	17.94	0.48	0.9954
0.001	22.30	557.20	-	0.9950	18.02	0.49	0.9954
0.01	20.75	558.47	-	0.9956	17.66	0.49	0.9956
0.1	16.20	550.69	-	0.9968	13.67	0.34	0.9968
0.2	12.93	635.15	-	0.9975	10.99	0.29	0.9975
0.3	10.02	615.11	1669.63	0.9980	8.48	0.22	0.9980
0.4	7.83	572.55	1426.41	0.9984	6.58	0.17	0.9984
0.5	6.39	513.75	1157.20	0.9987	5.42	0.14	0.9987
0.6	5.17	441.12	923.95	0.9989	4.40	0.12	0.9989
0.7	3.91	357.81	684.89	0.9992	3.31	0.10	0.9992
0.8	2.86	254.41	553.94	0.9994	2.42	0.07	0.9994
0.9	1.70	97.00	395.52	0.9996	1.44	0.05	0.9996

From Table 2.2, the active-set algorithms converge to the optimal solution in a few iterations, but the computational cost of each iteration is expensive, since the Hessian matrix is involved in the computation. Our method and the proximal method have a larger number of iterations, and the computational cost of each iteration is very low. Both methods have closed-form update solution. The operation on the Hessian is avoided in our SMO methods. Finally, we should mention that our SMO methods can obtain identical solutions to the active-set methods, which corroborates that the decomposition methods converge well. However, the proximal method can only obtain approximate results in many cases, because it is a first-order method that needs a large number of iterations to converge.

Additionally, we tested the prediction accuracies of our l_1QP and NNQP models for classifying the microarray gene profiles. We used our SMO methods in the optimizations. The mean accuracies of 10-fold cross-validation are given in Table 2.3. We can see that both models obtained very high accuracies. The NNQP model with $\lambda = 0$ obtained the

Table 2.2: Mean number of iterations of each sample when using 5356 samples as training set.

λ	l_1QP			NNQP	
	SMO	proximal	active-set	SMO	active-set
0	-	-	-	53033	29.37
0.0001	682153	29250	-	52994	29.36
0.001	55612	29427	-	52981	29.30
0.01	51438	29451	-	51439	28.23
0.1	40068	29302	-	40068	20.07
0.2	31962	34334	-	31962	15.43
0.3	24899	33212	13.30	24899	12.20
0.4	19348	30902	11.40	19348	9.80
0.5	15927	27752	9.40	15927	8.22
0.6	12853	23834	7.80	12853	6.88
0.7	9680	19236	6.00	9680	5.51
0.8	7092	13144	5.00	7092	4.30
0.9	4210	5152	3.80	4210	3.09

best result on this data. This observation suggests that we need to try the easier NNQP model first before resorting to l_1QP model, when applying sparse coding techniques in other problems, particularly in classification.

Table 2.3: Mean prediction accuracies of the l_1QP and NNQP models using 10-fold cross-validation.

λ	l_1QP	NNQP
0	-	0.9762
0.5	0.9727	0.9727
0.9	0.9654	0.9654

6 Bayesian Dictionary Learning [9]

We pursue our dictionary-learning-based approach for biological data, based on the following two motivations. First, the sparse-coding-only approach, which places all training samples

[9] This section is based on our publications [3], [17], and [34].

in the dictionary, is a lazy learning, thus the optimization can be slow for large training set. Therefore, learning a concise dictionary is more efficient for future real-time applications. Second, dictionary learning can capture hidden key factors which correspond to biological pathways, and the classification performance may hence be improved. In the following, we first give the dictionary learning models using Gaussian prior and uniform prior, respectively. Next, we give the classification method based on dictionary learning. We then address the generic optimization framework of dictionary learning. Finally, we show that the kernel versions of our dictionary learning models and classification approach can be easily obtained.

6.1 Dictionary Learning Models

Now we give our dictionary learning models using Gaussian prior and uniform prior over the dictionary atoms, respectively. Both priors aims to get rid off the arbitrary scale interchange between dictionary and coefficient. Suppose $\boldsymbol{D}_{m\times n}$ is the data of n training instances, and the dictionary $\boldsymbol{A}$ to be learned has k atoms. If the Gaussian prior in Equation (2.5) is used on the dictionary atom, our dictionary learning models of l_1LS, NNLS, and l_1NNLS are expressed as follow, respectively:

$$l_1LS: \min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) + \lambda\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1, \tag{2.66}$$

$$\begin{aligned} NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) \\ \text{s.t. } & \boldsymbol{Y}\geq 0, \end{aligned} \tag{2.67}$$

and

$$\begin{aligned} l_1NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) + \sum_{i=1}^{n}\boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y}_i \\ \text{s.t. } & \boldsymbol{Y}\geq 0. \end{aligned} \tag{2.68}$$

The strength of the Gaussian prior based dictionary learning is that, it is flexible to control the scales of dictionary atoms by tuning α. However, l_1LS and l_1NNLS have two model parameters (that is α and λ), which increase the model selection burden in practice.

Alternatively, in order to eliminate the parameter α, we design an uniform prior over

the dictionary which is expressed as

$$p(\boldsymbol{a}_i) = \begin{cases} \theta & \text{if } \|\boldsymbol{a_i}\|_{\boldsymbol{2}} = 1, \\ 0 & \text{otherwise,} \end{cases} \tag{2.69}$$

where $\theta \in (0, 1)$ is a constant. That is, the feasible region of the dictionary atoms is a hypersphere centered at the origin with unit radius, and all the feasible atoms have equal probability. The corresponding dictionary learning models are given in the following equations, respectively:

$$\begin{aligned} l_1LS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \lambda \sum_{i=1}^{n} \|\boldsymbol{y}_i\|_1 \\ \text{s.t. } & \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1, \cdots, k, \end{aligned} \tag{2.70}$$

$$\begin{aligned} NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 \\ \text{s.t. } & \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1, \cdots, k; \quad \boldsymbol{Y} \geq 0, \end{aligned} \tag{2.71}$$

and

$$\begin{aligned} l_1NNLS: \min_{\boldsymbol{A},\boldsymbol{Y}} & \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{n} \boldsymbol{\lambda}^{\mathrm{T}}\boldsymbol{y}_i \\ \text{s.t. } & \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{a}_i = 1, \quad i = 1, \cdots, k; \quad \boldsymbol{Y} \geq 0. \end{aligned} \tag{2.72}$$

6.2 A Generic Optimization Framework for Dictionary Learning

We devise a block-coordinate-descent based algorithms for the optimization of the above six models. The main idea is that $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{Y}$ are updated alternatingly. In a step, $\boldsymbol{Y}$ is fixed, and the inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, rather than $\boldsymbol{A}$ itself, is updated; in the next step, $\boldsymbol{Y}$ is updated while fixing $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ (a sparse coding procedure). The above procedure is repeated until the termination conditions are satisfied.

Now, we show that $\boldsymbol{A}$ can be analytically obtained. For normal prior over dictionary atoms, the optimization of finding $\boldsymbol{A}$ in Equations (2.66), (2.67), and (2.68) is to solve

$$\min_{\boldsymbol{A}} f(\boldsymbol{A}) = \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}). \tag{2.73}$$

Taking the derivative with respect to $\boldsymbol{A}$ and setting it to zero, we have

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}} + \alpha\boldsymbol{A} = 0. \tag{2.74}$$

We hence have

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\ddagger}, \tag{2.75}$$

where $\boldsymbol{Y}^{\ddagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha\boldsymbol{I})^{-1}$. The inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ can thus be updated by

$$\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}. \tag{2.76}$$

We also can compute $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ by

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}. \tag{2.77}$$

For the uniform prior as in Equation (2.69), updating unnormalized $\boldsymbol{A}$ while fixing $\boldsymbol{Y}$ in Equations (2.70), (2.71), and (2.72) is to solve the generalized least squares:

$$\min_{\boldsymbol{A}} f(\boldsymbol{A}) = \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2. \tag{2.78}$$

Taking the derivative with respect to $\boldsymbol{A}$ and setting it to zero, we have

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\dagger}, \tag{2.79}$$

where $\boldsymbol{Y}^{\dagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1}$. The inner products of $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ are computed similarly as for the Gaussian prior. The normalization of $\boldsymbol{R}$ is straightforward. We have $\boldsymbol{R} = \boldsymbol{R}./\sqrt{\mathrm{diag}(\boldsymbol{R})\mathrm{diag}(\boldsymbol{R})^{\mathrm{T}}}$, where $./$ and $\sqrt{\bullet}$ are element-wise operators. Learning the inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ instead of $\boldsymbol{A}$ has the benefits of dimension-free computation and kernelization.

Fixing $\boldsymbol{A}$, $\boldsymbol{Y}$ can be obtained via our active-set method described in Section 4.6. Recall that the sparse coding only requires the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. As shown above, we find that updating $\boldsymbol{Y}$ only needs its previous value and the inner product between training instances.

Due to the above derivation, we have the framework of solving our dictionary learning models as illustrated in Algorithm 2.5.

Algorithm 2.5 Generic Dictionary Learning Framework

Input: $\boldsymbol{K} = \boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}$, dictionary size k, λ
Output: $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{Y}$

initialize $\boldsymbol{Y}$ and $\boldsymbol{R} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ randomly;
$r_{prev} = Inf$; {previous residual}

for $i = 1 : maxIter$ **do**
  Update $\boldsymbol{Y}$ by solving the active-set based l_1LS, NNLS, or l_1NNLS sparse coding algorithms;

  if Gaussian prior over $\boldsymbol{A}$ **then**
    Update $\boldsymbol{R} = \boldsymbol{Y}^{\ddagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}$;
  end if

  if uniform prior over $\boldsymbol{A}$ **then**
    Update $\boldsymbol{R} = \boldsymbol{Y}^{\dagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\dagger}$;
    Normalize $\boldsymbol{R}$ by $\boldsymbol{R} = \boldsymbol{R}./\sqrt{\mathrm{diag}(\boldsymbol{R})\mathrm{diag}(\boldsymbol{R})^{\mathrm{T}}}$;
  end if

  if $i == maxIter$ or $i \mod l == 0$ **then**
    {check every l iterations}
    $r_{cur} = f(\boldsymbol{A}, \boldsymbol{Y})$; {current residual of a dictionary learning model}
    if $r_{prev} - r_{cur} \leq \epsilon$ or $r_{cur} \leq \epsilon$ **then**
      Break;
    end if
    $r_{prev} = r_{cur}$;
  end if
end for

6.3 Classification Approach Based on Dictionary Learning

Now, we present the dictionary-learning based classification approach in Algorithm 2.6. The dictionary learning in the training step should be consistent with the sparse coding in the prediction step. As discussed in the previous section, the sparse coding in the prediction step needs the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$. Actually, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$ is either $\boldsymbol{Y}^{\ddagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{B}$ or $\boldsymbol{Y}^{\dagger\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{B}$.

Algorithm 2.6 *Dictionary-Learning Based Classification*

Input: $\boldsymbol{D}_{m\times n}$: n training instances, $\boldsymbol{c}$ the class labels, $\boldsymbol{B}_{m\times p}$: p new instances, k: dictionary size

Output: $\boldsymbol{p}$: the predicted class labels of the p new instances

{**Training Step:**}

1. Normalize each training instance to have unit l_2 norm.
2. Learn dictionary inner product $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and sparse coefficient matrix $\boldsymbol{Y}$ of training instances by Algorithm 2.5.
3. Train a classifier $f(\theta)$ with parameter θ using $\boldsymbol{Y}$ (in the feature space spanned by columns of $\boldsymbol{A}$).

{**Prediction Step:**}

1. Normalize each new instance to have unit l_2 norm.
2. Obtain the sparse coefficient matrix $\boldsymbol{X}$ of the new instances by solving Equation (2.42), or (2.43).
3. Predict the class labels of $\boldsymbol{X}$ using the classifier $f(\theta)$ learned in the training phase.

6.4 Kernel Extensions

For Gaussian dictionary prior, the l_1LS based kernel dictionary learning and sparse coding are expressed in the following, respectively:

$$\min_{\boldsymbol{A}_\phi,\boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D}) - \boldsymbol{A}_\phi\boldsymbol{Y}\|_F^2 + \frac{\alpha}{2}\mathrm{trace}(\boldsymbol{A}_\phi^{\mathrm{T}}\boldsymbol{A}_\phi) + \lambda\|\boldsymbol{Y}\|_1, \tag{2.80}$$

$$\min_{\boldsymbol{X}} \frac{1}{2}\|\phi(\boldsymbol{B}) - \boldsymbol{A}_\phi\boldsymbol{X}\|_F^2 + \lambda\|\boldsymbol{X}\|_1, \tag{2.81}$$

where $\phi(\bullet)$ is a mapping function. Equations (2.67), (2.68), (2.70), (2.71), (2.72) and their sparse coding models can be kernelized analogously. As we have mentioned already, the

optimizations of the six dictionary learning models only involve inner products of instances. Thus, we can easily obtain their kernel extensions by replacing the inner products with kernel matrices. Hereafter, if dictionary learning is employed in sparse representation, then prefix "DL" is used before "l_1LS", "NNLS", and "l_1NNLS". If kernel function other than the linear kernel is used in dictionary learning, then prefix "KDL" is added before them.

6.5 Computational Experiments

Two high-throughput biological data, including a microarray gene expression data set and a protein mass spectrometry data set, are used to test the performance of our dictionary learning methods for dimensionality reduction. The microarray data set is a collection of gene expression profiles of breast cancer subtypes [31]. This data set includes 158 tumor samples from five subtypes measured on 13582 genes. The mass spectrometry data set is composed of 332 samples from normal class and prostate cancer class [32]. Each sample has 15154 features, that is the mass-to-charge ratios. The performance is measured by accuracy and running time in seconds.

The performance of various dictionary learning models with linear and RBF kernels were investigated on both Breast and Prostate data sets. The Gaussian-prior based and uniform-prior based dictionary learning models were also compared. Again, our active-set dictionary learning method was compared with the interior-point [23] and proximal [25] methods. The semi-NMF based on multiplicative update rules [35] is also included in the comparison. As in the experiment of sparse coding in Section 5, four-fold cross-validation was used. All methods ran on the same splits of training and test sets. We performed 20 runs of cross-validation for reliable comparison. After feature extraction by using dictionary learning on the training set, the linear SVM classifier was trained on the reduced training set and was used to predict the class labels of test instances.

In Figure 2.3, we show the mean accuracy and standard deviation of 20 results for each method. First, we can see that the models with Gaussian prior on dictionary atoms obtained similar accuracies as the uniform prior. Second, with the comparison to sparse coding methods on Breast data as given Figure 2.1, we can see that dictionary learning increases the prediction accuracy. Third, from the comparison of Figures 2.3 and 2.1, we find that the dictionary learning based methods – DL-NNLS and DL-l_1NNLS, obtained similar accuracies as the sparse coding methods – NNLS and l_1NNLS. This convinces us that dictionary learning is a promising feature extraction technique for high-dimensional biological data. On Prostate data, we can also find that the accuracy obtained by DL-l_1LS is slightly lower than l_1LS. This is may be because the dictionary learning is unsupervised. Fourth, using

the model parameters, DL-l_1LS using active-set algorithm obtained higher accuracy than DL-l_1LS-IP and DL-l_1LS-PX on Prostate data. The accuracy of DL-l_1LS is also slightly higher than that of DL-l_1LS-IP on Breast data. Furthermore, the non-negative DL-NNLS yielded the same performance as the well-known semi-NMF, while further corroborating the satisfactory performance of our dictionary learning framework. Finally, the kernel dictionary learning models achieved similar performance as their linear counterparts. We believe that the accuracy could be further improved by a suitably selected kernel.

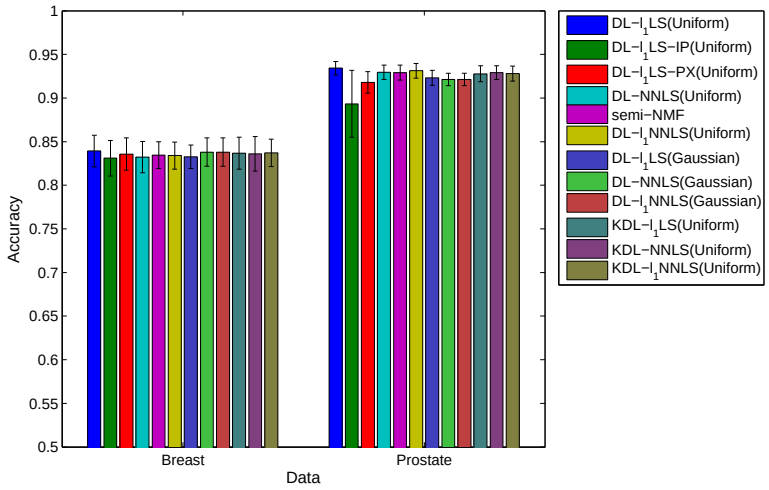


Figure 2.3: Accuracies of dictionary learning approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation.

We compare the mean computing time of all the feature extraction methods in Figure 2.4. First, we can see that DL-l_1LS using active-set algorithm is much more efficient than DL-l_1LS-IP, DL-l_1LS-PX, and semi-NMF using multiplicative update rules. Second, the non-negative dictionary learning models are more efficient than the l_1-regularized models. Therefore as in the sparse coding method, priority should be given to the non-negative models when attempting to use dictionary learning in an application.

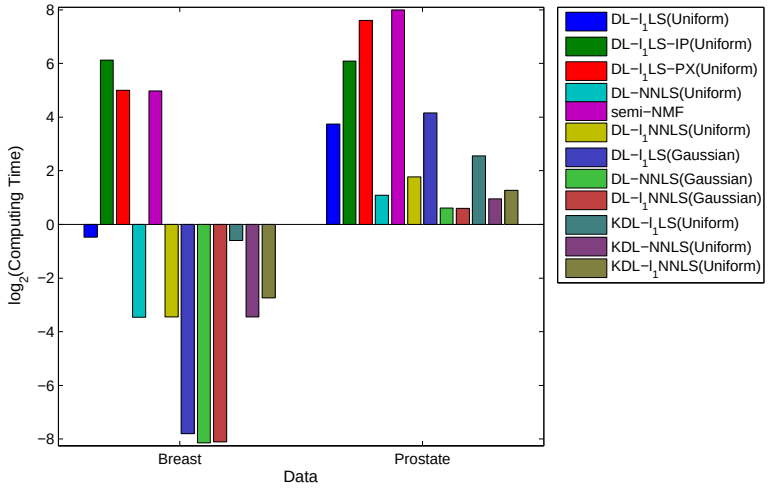


Figure 2.4: Computing time of dictionary learning approaches. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend. In the legend, the Gaussian and uniform priors are indicated in the corresponding parentheses. IP and PX are the abbreviations of the interior-point and proximal methods, respectively. The rest dictionary learning models use active-set method without explicit notation.

7 Versatile Sparse Matrix Factorization [10]

We have proposed a generic dictionary learning framework in Section 6.1. Now, we present a more general dictionary learning framework, named *versatile sparse matrix factorization* (VSMF).

7.1 Versatile Sparse Matrix Factorization Model

Our VSMF model can be expressed in the following equation:

$$\begin{aligned} \min_{\boldsymbol{A},\boldsymbol{Y}} f(\boldsymbol{A},\boldsymbol{Y}) &= \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{k}\left(\frac{\alpha_2}{2}\|\boldsymbol{a}_i\|_2^2 + \alpha_1\|\boldsymbol{a}_i\|_1\right) \\ &\quad + \sum_{i=1}^{n}\left(\frac{\lambda_2}{2}\|\boldsymbol{y}_i\|_2^2 + \lambda_1\|\boldsymbol{y}_i\|_1\right) \\ \text{s.t.} &\begin{cases} \text{if } t_1 = 1 \quad \boldsymbol{A} \geq 0 \\ \text{if } t_2 = 1 \quad \boldsymbol{Y} \geq 0 \end{cases}, \end{aligned} \tag{2.82}$$

[10] This section is based our publication [36].

where, parameter $\alpha_1 \geq 0$ controls the sparsity of the basis vectors; parameter $\alpha_2 \geq 0$ controls the smoothness and scale of the basis vectors; parameter $\lambda_1 \geq 0$ controls the sparsity of the coefficient vectors; parameter $\lambda_2 \geq 0$ controls the smoothness of the coefficient vectors; and parameters t_1 and t_2 are boolean variables (0: false, 1: true) that indicate if non-negativity should be enforced on $\boldsymbol{A}$ and $\boldsymbol{Y}$, respectively.

One advantage of VSMF is that both l_1 and l_2-norms can be used on both basis matrix and coefficient matrix. In VSMF, l_1-norms are used to induce sparse basis vectors and coefficient vectors. However, the drawback of l_1-norm is that correlated variables may not be simultaneously non-zero in the induced sparse result. This is because l_1-norm is able to produce sparse but non-smooth result. It is known that l_2-norm is able to obtain smooth but not sparse result. It has been demonstrated that correlated variables can be selected or removed simultaneously via combining both norms [37]. In addition to the smoothness of l_2-norm, another benefit of l_2-norm is that the scale of each vector can be restricted. This can avoid the scale interchange between the basis matrix and the coefficient matrix. Another advantage of VSMF is that the non-negativity constraint can be switched off/on for either basis matrix or coefficient matrix. If the training data are non-negative, it is usually necessary that the basis matrix should be non-negative as well. In some situations, non-negativity is also needed on the coefficient matrix for better performance and better interpretability of results.

For the convenience of discussion, we summarize the existing sparse matrix factorization models in Table 2.4. It is impossible to enumerate all existing works in this direction. Therefore, all models mentioned in this table are the most representative ones. The training data $\boldsymbol{D}$ must be non-negative for the standard NMF and sparse NMF. For sparse NMF, α and λ are two non-negative parameters. For kernel NMF and DL-l_1LS, $\phi(\cdot)$ is a function that maps the training samples into a high-dimensional feature space. $\phi(\boldsymbol{D})$ is the training samples in this feature space. $\boldsymbol{A}_\phi$ is the basis matrix in this feature space. It can be easily seen that the standard NMF, semi-NMF, and sparse-NMF are special cases of VSMF. If $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$ and $t_1 = t_2 = 1$, VSMF is reduced to the standard NMF proposed in [12]. If $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$ and $t_1 = 0$ and $t_2 = 1$, then VSMF becomes semi-NMF proposed in [35]. If $\alpha_1 = \lambda_2 = 0$, $\alpha_2, \lambda_1 \neq 0$, and $t_1 = t_2 = 1$, then VSMF is equivalent to the sparse-NMF proposed in [38]. When α_1 is set to zero, VSMF can be kernelized (see Section 6.1 and [3]).

Sparse matrix factorization is a low-rank approximation problem. The number of ranks, that is k, is crucial for a good performance of an analysis. Selecting k is still an open problem in both statistical inference and machine learning. We propose an adaptive rank

Table 2.4: The existing NMF and SR models.

NMF/SR	Equation
Standard NMF [12]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2$ s.t. $\boldsymbol{A},\boldsymbol{Y}\geq 0$
Semi-NMF [35]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2$ s.t. $\boldsymbol{Y}\geq 0$
Sparse NMF [38]	$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2+\frac{\alpha}{2}\sum_{i=1}^{k}\|\boldsymbol{a}_i\|_2^2+\frac{\lambda}{2}\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1^2$ s.t. $\boldsymbol{A},\boldsymbol{Y}\geq 0$
Kernel NMF [39, 3]	$\min_{\boldsymbol{A}_\phi,\boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D})-\boldsymbol{A}_\phi\boldsymbol{Y}\|_F^2+\frac{\alpha}{2}\sum_{i=1}^{k}\|\phi(\boldsymbol{a}_i)\|_2^2+\frac{\lambda}{2}\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1$ s.t. $\boldsymbol{Y}\geq 0$
l_1-SR [3]	$\min_{\boldsymbol{A}_\phi,\boldsymbol{Y}} \frac{1}{2}\|\phi(\boldsymbol{D})-\boldsymbol{A}_\phi\boldsymbol{Y}\|_F^2+\frac{\alpha}{2}\sum_{i=1}^{k}\|\phi(\boldsymbol{a}_i)\|_2^2+\frac{\lambda}{2}\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1$

selection method for VSMF. We base our idea on the sparsity of columns of $\boldsymbol{A}$ and $\boldsymbol{Y}$. We first set k to a relatively large integer. During the optimization of VSMF, if a column of $\boldsymbol{A}$ or a row of $\boldsymbol{Y}$ is null due to the sparsity controlled by the corresponding parameters, then both of the column of $\boldsymbol{A}$ and the row of $\boldsymbol{Y}$ corresponding to this null factor are removed. Therefore, k is reduced. When the optimization terminates, we can obtain the correct k corresponding to the current sparsity controlling parameters.

7.2 Optimization

Like most of NMF and SR models, the VSMF model is non-convex (a model is said to be convex if and only if both of the objective and constraints are convex). The most popular scheme to optimize these models are the block-coordinate descent method [29]. The basic idea of this scheme is in the following. $\boldsymbol{A}$ and $\boldsymbol{Y}$ are updated iteratively and alternatingly. In each iteration, $\boldsymbol{A}$ is updated while keeping $\boldsymbol{Y}$ fixed; then $\boldsymbol{A}$ is fixed and $\boldsymbol{Y}$ is updated. Based on this scheme, we devise the multiplicative update rules and active-set algorithms for VSMF. These two algorithms are given below.

Multiplicative Update Rules

If both $\boldsymbol{A}$ and $\boldsymbol{Y}$ are non-negative, we can equivalently rewrite $f(\boldsymbol{A},\boldsymbol{Y})$ in Equation (3.11) as

$$\frac{1}{2}\|\boldsymbol{D}-\boldsymbol{A}\boldsymbol{Y}\|_F^2+\frac{\alpha_2}{2}\mathrm{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})+\alpha_1\mathrm{trace}(\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A})+\frac{\lambda_2}{2}\mathrm{trace}(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y})+\lambda_1\mathrm{trace}(\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y}), \tag{2.83}$$

where $\boldsymbol{E}_1 \in \{1\}^{m\times k}$, and $\boldsymbol{E}_2 \in \{1\}^{k\times n}$. Fixing $\boldsymbol{A}$, and updating $\boldsymbol{Y}$ can hence be expressed as

$$\begin{aligned} &\min_{\boldsymbol{Y}} f(\boldsymbol{Y}) = \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\lambda_2}{2}\text{trace}(\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}) + \lambda_1\text{trace}(\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y}) \\ &\text{s.t. } \boldsymbol{Y} \geq 0. \end{aligned} \tag{2.84}$$

Similarly, fixing $\boldsymbol{Y}$, and updating $\boldsymbol{A}$ can be expressed as

$$\begin{aligned} &\min_{\boldsymbol{A}} f(\boldsymbol{A}) = \frac{1}{2}\|\boldsymbol{D} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\alpha_2}{2}\text{trace}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}) + \alpha_1\text{trace}(\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A}) \\ &\text{s.t. } \boldsymbol{A} \geq 0. \end{aligned} \tag{2.85}$$

We design the following multiplicative update rules for VSMF model in the case of $t_1 = t_2 = 1$:

$$\begin{cases} \boldsymbol{A} = \boldsymbol{A} * \frac{\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}+\alpha_2\boldsymbol{A}+\alpha_1} \\ \boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y}+\lambda_2\boldsymbol{Y}+\lambda_1} \end{cases}, \tag{2.86}$$

where $\boldsymbol{A} * \boldsymbol{B}$ and $\frac{\boldsymbol{A}}{\boldsymbol{B}}$ are element-wise multiplication and division between matrix $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively. This algorithm is a gradient-descent based method in essence. Both rules are derived in the following.

For Equation (2.84), the first-order update rule of $\boldsymbol{Y}$ should be generally

$$\boldsymbol{Y} = \boldsymbol{Y} - \boldsymbol{\eta}_2 * \frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}}. \tag{2.87}$$

where matrix $\boldsymbol{\eta}_2$ is the step size of the update rule. We take the derivative of $f(\boldsymbol{Y})$, in Equation (2.84), with respect to $\boldsymbol{Y}$:

$$\frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} + \lambda_2\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2. \tag{2.88}$$

And we let the step size $\boldsymbol{\eta}_2$ to be

$$\boldsymbol{\eta}_2 = \frac{\boldsymbol{Y}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \lambda_2\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2}. \tag{2.89}$$

Substituting Equations (2.88) and (2.89) into Equation (2.87), we have

$$\boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \lambda_2\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2}. \tag{2.90}$$

Similarly, for Equation (2.85), the first-order update rule of $\boldsymbol{A}$ should be generally

$$\boldsymbol{A} = \boldsymbol{A} - \boldsymbol{\eta}_1 * \frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}}. \tag{2.91}$$

We take the derivative of $f(\boldsymbol{A})$, in Equation (2.85), with respect to $\boldsymbol{A}$:

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{Y} + \alpha_2\boldsymbol{A} + \alpha_1\boldsymbol{E}_1. \tag{2.92}$$

And we let the step size to be

$$\boldsymbol{\eta}_1 = \frac{\boldsymbol{A}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{A} + \alpha_1\boldsymbol{E}_1}. \tag{2.93}$$

Substituting Equations (2.92) and (2.93) into Equation (2.91), we have

$$\boldsymbol{A} = \boldsymbol{A} * \frac{\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{A} + \alpha_1}. \tag{2.94}$$

If we let $\alpha_1 = \alpha_2 = \lambda_1 = \lambda_2 = 0$, then the update rules in Equations (2.86) becomes the update rules of the standard NMF [40]. We can find that enforcing sparsity and smoothness on both basis matrix and coefficient matrix does not increase the time-complexity. Finally, we have to reminder the readers that, the multiplicative update rule is the trade-off between the complexity of designing an algorithm and the precision. Its advantage is that it is easy to implement. However, it has two weaknesses as follows. First, it can be proven that the objective is non-increasing, but it is not guaranteed to converge to a stationary point [41]. Second, $\boldsymbol{A}$ and $\boldsymbol{Y}$ are usually initialized by random numbers. If a value of them becomes zero, it won't escape from it. The readers are advised to see [42] for a detailed discussion and a revised multiplicative update rule.

Active-Set Quadratic Programming

The multiplicative update rules above only work under the condition that both $\boldsymbol{A}$ and $\boldsymbol{Y}$ are non-negative. We devise active-set algorithms which allow us to relax the non-negativity constraints. We now show that when $t_1($ or $t_2) = 1$, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated by our active-set NNQP algorithm described in Algorithm 2.4 and Section 4.6; when $t_1($ or $t_2) = 0$, $\boldsymbol{A}$

(or $\boldsymbol{Y}$) can be updated by our active-set 1_1QP algorithm described in Algorithm 2.3 and Section 4.6.

If $t_2 = 1$, the objective in Equation (2.84) can be rewritten as:

$$\begin{aligned} f(\boldsymbol{Y}) &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{\lambda_2}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y} + \lambda_1\boldsymbol{E}_2^{\mathrm{T}}\boldsymbol{Y}) \\ &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I})\boldsymbol{Y} + (\lambda_1\boldsymbol{E}_2^{\mathrm{T}} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}) \\ &= \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{H}_2\boldsymbol{y}_i + \boldsymbol{g}_{(2)i}^{\mathrm{T}}\boldsymbol{y}_i + \frac{1}{2}\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_i, \end{aligned} \tag{2.95}$$

where $\boldsymbol{H}_2 = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I}$, and $\boldsymbol{g}_{(2)i} = \lambda_1 - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{d}_i$ and $\boldsymbol{G}_{(2)} = \lambda_1 - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. Therefore, we can see that updating non-negative $\boldsymbol{Y}$ is a multiple NNPQ problem. The parallel active-set algorithm for NNQP, proposed in Section 4.6, can be used to solve the problem in Equation (2.95).

If $t_2 = 0$, the objective of updating $\boldsymbol{Y}$ can be reformulated as:

$$\begin{aligned} f(\boldsymbol{Y}) &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y} + \frac{\lambda_2}{2}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{Y}) + \lambda_1\|\boldsymbol{Y}\|_1 \\ &= \text{trace}(\frac{1}{2}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I})\boldsymbol{Y} + (-\boldsymbol{D}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{Y} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}) + \lambda_1\|\boldsymbol{Y}\|_1 \\ &= \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}\boldsymbol{H}_2\boldsymbol{y}_i + \boldsymbol{g}_{(2)i}^{\mathrm{T}}\boldsymbol{y}_i + \lambda_1\|\boldsymbol{y}_i\|_1 + \frac{1}{2}\boldsymbol{d}_i^{\mathrm{T}}\boldsymbol{d}_i, \end{aligned} \tag{2.96}$$

where $\boldsymbol{H}_2 = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I}$, and $\boldsymbol{g}_{(2)i} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{d}_i$ and $\boldsymbol{G}_{(2)} = -\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. This is a l_1QP problem which can be solved by our active-set l_1QP algorithm proposed in Section 4.6 and [3].

Similarly, if $t_1 = 1$, $f(\boldsymbol{A})$ in Equation (2.84) can be expressed as

$$\begin{aligned} f(\boldsymbol{A}) &= \text{trace}(\frac{1}{2}\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{\alpha_2}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} + \alpha_1\boldsymbol{E}_1^{\mathrm{T}}\boldsymbol{A}) \\ &= \text{trace}(\frac{1}{2}\boldsymbol{A}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I})\boldsymbol{A}^{\mathrm{T}} + (\alpha_1\boldsymbol{E}_1^{\mathrm{T}} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}})\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}) \\ &= \sum_{i=1}^{m} \frac{1}{2}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{H}_1\boldsymbol{w}_i + \boldsymbol{g}_{(1)i}^{\mathrm{T}}\boldsymbol{w}_i + \frac{1}{2}\boldsymbol{D}_{i,:}(\boldsymbol{D}^{\mathrm{T}})_{:,i}, \end{aligned} \tag{2.97}$$

where $\boldsymbol{W} = \boldsymbol{A}^{\mathrm{T}}$, $\boldsymbol{H}_1 = \boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I}$, $\boldsymbol{g}_{(1)i} = \alpha_1 - \boldsymbol{Y}(\boldsymbol{D}^{\mathrm{T}})_{:,i}$ and $\boldsymbol{G}_{(1)} = \alpha_1 - \boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$. Again, it can be seen that this problem is also a NNQP problem which can be solved by the active-set algorithm in Section 4.6 and [3].

If $t_1 = 0$, the objective of updating $\boldsymbol{A}$ can be written as

$$\begin{aligned} f(\boldsymbol{A}) &= \text{trace}(\frac{1}{2}\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D} - \boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}} + \frac{\alpha_2}{2}\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}}) + \alpha_1\|\boldsymbol{A}\|_1 \\ &= \text{trace}(\frac{1}{2}\boldsymbol{A}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I})\boldsymbol{A}^{\mathrm{T}} + (-\boldsymbol{D}\boldsymbol{Y}^{\mathrm{T}})\boldsymbol{A}^{\mathrm{T}} + \frac{1}{2}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}) + \alpha_1\|\boldsymbol{A}^{\mathrm{T}}\|_1 \\ &= \sum_{i=1}^{m} \frac{1}{2}\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{H}_{\boldsymbol{1}}\boldsymbol{w}_i + \boldsymbol{g}_{(1)i}^{\mathrm{T}}\boldsymbol{w}_i + \alpha_1\|\boldsymbol{w}_i\|_1 + \frac{1}{2}\boldsymbol{D}_{i,:}(\boldsymbol{D}^{\mathrm{T}})_{:,i}, \end{aligned} \tag{2.98}$$

where $\boldsymbol{W} = \boldsymbol{A}^{\mathrm{T}}$, $\boldsymbol{H}_{\boldsymbol{1}} = \boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I}$, $\boldsymbol{g}_{(1)i} = -\boldsymbol{Y}(\boldsymbol{D}^{\mathrm{T}})_{:,i}$ and $\boldsymbol{G}_{(1)} = -\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$. This is also a l_1QP problem that can be solved by our active-set l_1QP algorithm proposed in Section 4.6 and [3].

Analytical Solutions and Kernelization

If $t_2 = 0$ and $\lambda_1 = 0$, from $\frac{\partial f(\boldsymbol{Y})}{\partial \boldsymbol{Y}} = 0$, $\boldsymbol{Y}$ can be updated analytically, that is,

$$\boldsymbol{Y} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \lambda_2\boldsymbol{I})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}. \tag{2.99}$$

From the previous section, we can see that only $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ are required to update $\boldsymbol{A}$. According to Equation (2.99), $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ can be expressed as

$$\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}(\boldsymbol{A}^{\ddagger})^{\mathrm{T}}. \tag{2.100}$$

$$\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}} = \boldsymbol{A}^{\ddagger}\boldsymbol{D}\boldsymbol{D}^{\mathrm{T}}. \tag{2.101}$$

We can see that in this situation, updating $\boldsymbol{A}$ only requires the previous value of $\boldsymbol{A}$ and the inner products of the rows of $\boldsymbol{D}$.

Similarly, if $t_1 = 0$ and $\alpha_1 = 0$, $\boldsymbol{A}$ can be updated analytically, that is,

$$\boldsymbol{A} = \boldsymbol{D}\boldsymbol{Y}^{\ddagger}, \tag{2.102}$$

where $\boldsymbol{Y}^{\ddagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}} + \alpha_2\boldsymbol{I})^{-1}$. From the previous section, we know that updating $\boldsymbol{Y}$ only requires the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$. They can be updated by the following equations:

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}\boldsymbol{Y}^{\ddagger}. \tag{2.103}$$

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D} = (\boldsymbol{Y}^{\ddagger})^{\mathrm{T}}\boldsymbol{D}^{\mathrm{T}}\boldsymbol{D}. \tag{2.104}$$

Due to the analytical solution of $\boldsymbol{A}$, updating $\boldsymbol{Y}$ only requires the previous value of $\boldsymbol{Y}$ and the inner products of columns of $\boldsymbol{D}$.

These analytical solutions have two advantages. First, the corresponding matrix can be easily updated without resorting to any numerical solver. Second, we can see that only the inner products are needed to update $\boldsymbol{Y}$ (or $\boldsymbol{A}$), when $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be analytically obtained. Using this property, we can obtain the kernel version of VSMF, which are described in the following. In sparse representation, at least one matrix among $\boldsymbol{A}$ and $\boldsymbol{Y}$ must be sparse. That is, the analytical solutions in Equations (2.99) and (2.102) can not be used simultaneously. In practice, if each column of the training data $\boldsymbol{D}$ is the object to be mapped in high-dimensional feature space, we can analytically update $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ (or the corresponding kernel version $k(\boldsymbol{A},\boldsymbol{A}) = (\phi(\boldsymbol{A}))^{\mathrm{T}}\phi(\boldsymbol{A})$ where $k(\cdot,\cdot)$ is a kernel function corresponding to $\phi(\cdot)$) and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{D}$ (or $k(\boldsymbol{A},\boldsymbol{D}) = (\phi(\boldsymbol{A}))^{\mathrm{T}}\phi(\boldsymbol{D})$), and then update $\boldsymbol{Y}$ via a numerical solver described in the previous section. This leads to the kernel sparse representation proposed in [3]. Alternatively, if each row of $\boldsymbol{D}$ is the object to be mapped in high-dimensional feature space, $\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}$ and $\boldsymbol{Y}\boldsymbol{D}^{\mathrm{T}}$ should be updated analytically, then $\boldsymbol{A}$ is updated by a solver given in the previous section. This leads to an alternative kernel sparse representation model.

7.3 Computational Experiment

Sparse matrix factorization has a wide ranges of applications in biological data analysis. Technically speaking, these applications are based on clustering, biclustering, feature extraction, classification, and feature selection. In this study, we give three examples to show that promising performance can be obtained by VSMF for feature extraction, feature selection, and biological process identification. For other applications of NMF, please refer to [11].

Feature Extraction and Classification

NMF is a successful feature extraction method in bioinformatics [43]. Dimensionality reduction including feature extraction and feature selection is an important step in classification. We compared the performance of our VSMF (for feature extraction) with NMF on a popular microarray gene expression data – Colon [44]. This data set has 2000 genes (features) and 62 samples. There are two classes in this data set. Each sample is normalized to have unit l_2-norm. We employed four-fold cross-validation to split the whole data into training and test sets. For each split, features were extracted by NMF or VSMF from the training

set. The *nearest neighbor* (NN) classifier was used to predict the class labels of the test set. Four-fold cross-validation was repeated for 20 times. We initialized $k = 8$, thus the actual value of k, after calling VSMF, should be less than or equal to 8. *Radial basis function* (RBF) is used in the kernel VSMF. We set the kernel parameter $\sigma = 2^0$. The mean accuracy, *standard deviation* (STD), computing time, and parameter setting are given in Table 2.5. The standard NMF obtained a mean accuracy of 0.7645, while the linear VSMF yielded 0.7919. The highest accuracy, 0.7944, is obtained by the kernel VSMF. The kernel VSMF only took 1.3346 seconds, which is faster than the linear VSMF and NMF, because the analytical solution of $\boldsymbol{A}$ can be computed for kernel VSMF. We treated this comparison as a demonstration that tuning the parameters of VSMF may obtain better accuracy than NMF. VSMF can be used for many other types of high-throughput data such as copy number profiles and mass spectrometry data.

Table 2.5: Classification performance of VSMF compared to the standard NMF. The time is measure by stopwatch timer (the `tic` and `toc` functions in MATLAB) in seconds.

Method	Accuracy (STD)	Time	Parameters
NN	0.7742(0.0260)	0.0137	-
NMF+NN	0.7645(0.0344)	4.3310	-
Linear VSMF+NN	0.7919(0.0353)	3.1868	$\alpha_2 = 2^{-3}, \lambda_1 = 2^{-6}, t_1 = t_2 = 1$
Kernel VSMF+NN	0.7944(0.0438)	1.3346	$\alpha_2 = 2^{-3}, \lambda_1 = 2^{-6}, t_1 = t_2 = 1, \sigma = 2^0$

Feature Selection

VSMF can be applied to select relevant features. The basic idea is to make the basis vectors sparse, and then select features that vary dramatically among the basis vectors. In our current study of gene selection (genes are features), we use the following strategy on the sparse basis matrix $\boldsymbol{A}$. For the i-th row (that is the i-th gene), We denote $\boldsymbol{g}_i = \boldsymbol{A}_{i,:}$. If the maximum value in $\boldsymbol{g}_i$ is $\theta = 10^4$ times greater than the remaining values in $\boldsymbol{g}_i$, then we select this gene, otherwise we discard it. We tested this VSMF-based feature selection method on a microarray breast tumor data set which have 13582 genes and 158 samples from five classes [31]. The data were normalized so that each gene has mean zero and standard deviation (STD) one. We used the following parameters of VSMF: $\alpha_1 = 2^4$, $\alpha_2 = 2^0$, $\lambda_1 = 0$, $\lambda_2 = 2^0$, $t_1 = 0$, and $t_2 = 1$. The value of k was initialized to 5. The selected genes were validated by classification performance. We employed 20 runs of four-fold cross-validation. For each split of training and test sets, genes were selected using the training set. On the reduced training

set, a linear SVM was trained in order to predict the class labels of the corresponding test set. When using all genes to training SVM, we obtained a mean accuracy of 0.8250 with STD 0.0201. When applying the VSMF-based gene selection, we achieved a mean accuracy of 0.8271 with STD 0.0174. We can see that SVM using our gene selection strategy can obtain similar performance with that of using all genes.

7.4 Biological Process Identification

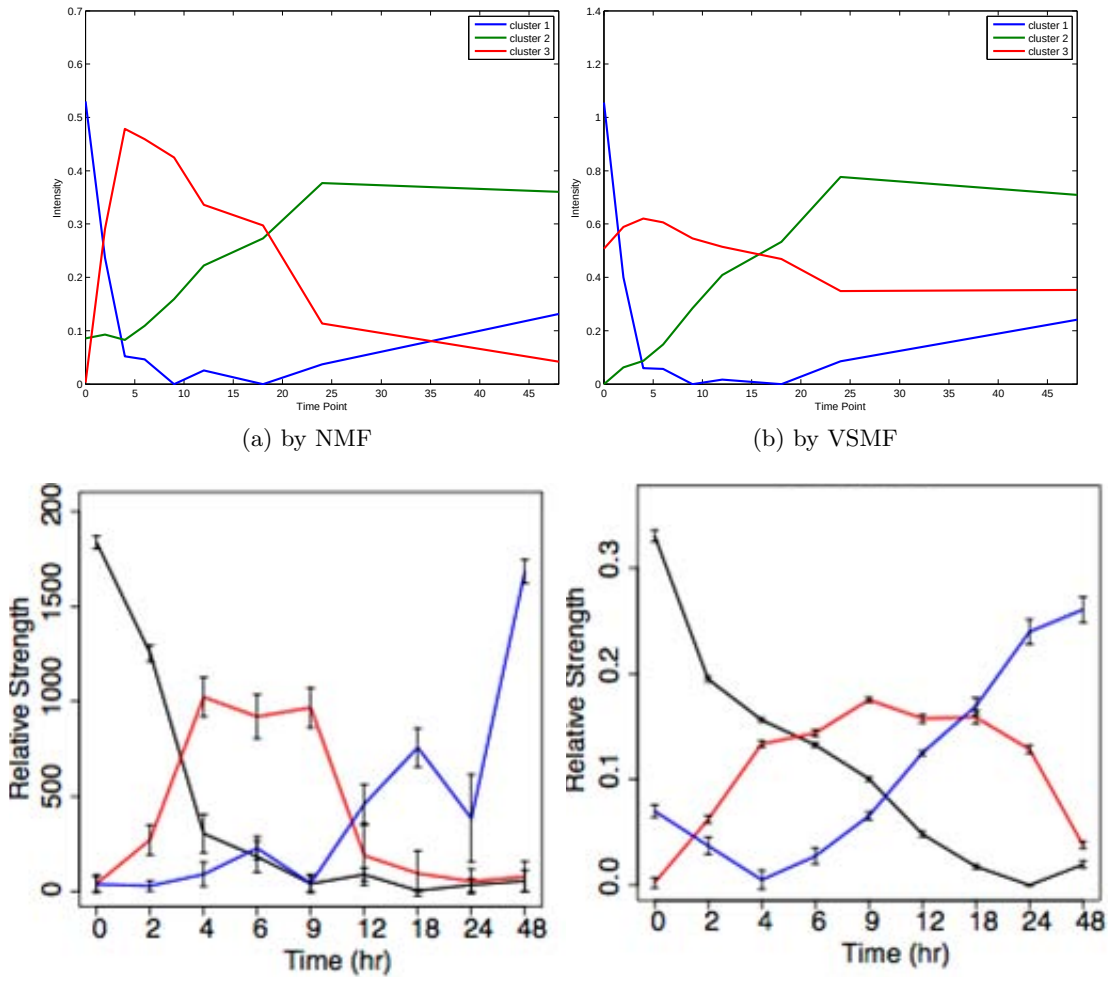
NMF has been applied on either gene-sample microarray data or time-series microarray data in order to identify potential biological processes [45, 38, 46, 1]. A biological process is defined as a series (or collection) of molecular functions (or events) (see `http://www.geneontology.org`). In our experiment, we run our VSMF on the *Gastrointestinal stromal tumor* (GIST) time-series data [1] to show that VSMF can smooth biological processes compared with the result obtained by the standard NMF. This data set was obtained after the treatment of imatinib mesylate. It has 1336 genes and 9 time points. Each gene time-series is normalized to have unit l_2-norm. The smoothness is controlled by parameter α_2. We set the parameters of VSMF to $\alpha_2 = 2^{-2}$, $\lambda_1 = 2^{-8}$, $\alpha_1 = \lambda_2 = 0$, and $t_1 = t_2 = 1$. The number of factors k was set to 3. The basis vectors of NMF and VSMF are shown in Figures 2.5a and 2.5b, respectively. The results obtained by the NMF and *coordinated gene activity in pattern sets* (CoGAPS, a variant of NMF optimized by Markov chain Monte Carlo sampling) of Ochs *et al.* are shown in Figures 2.5c and 2.5d. When comparing the plots in Figure 2.5, please note that the horizontal axis on the top are different from that at the bottom. We can see that all methods are able to reconstruct the falling, rising, and transient patterns identified in [1]. The patterns obtained by VSMF and CoGAPS are smoother than that of the standard NMF.

8 Supervised Dictionary Learning [11]

When applying sparse representation to classifying a data set with a large number of classes and samples, we have the following challenges:

1. As instance-based learning, the sparse coding methods without dictionary learning are able to classify complex data, but become very slow as the number of samples increases dramatically.

[11] This section is based on our publication [47].



(a) by NMF (b) by VSMF

(c) by NMF [1] (d) by CoGAPS [1]

Figure 2.5: The biological processes identified by our implementations of the standard NMF (a) and VSMF (b), and by Ochs *et al.*'s implementations of the standard NMF (c) and CoGAPS (d) [1].

2. The unsupervised dictionary learning learns a generative model and maintains a small dictionary, however it does not learn well the representations of complex multi-class data.

Taking the advantages of both sparse coding and dictionary learning, we propose the concept of *sub-dictionary learning*. The work of *meta-sample based sparse representation classification* (MSRC) [48] is in fact a specific case of sub-dictionary learning. MSRC employs SVD to learn the sub-dictionaries. The sparse coding of a new sample is obtained

through solving Equation (2.9). After that, the NS rule is used to predict the class label. Sound performance of MSRC was reported over microarray data. NMF was also tried to learn sub-dictionaries, but it was claimed, in [48], that its performance is worse than that of using SVD.

We find the sub-dictionary learning principle has the following advantages: i) this is eager learning, hence is applicable in real-time prediction; ii) dictionary learning over data of large sample size and many classes can be solved through a divide-and-conquer scheme; iii) the learning on each class is independent from others, therefore the classifier is easy to upgrade when there are some new training samples from some of the classes; and iv) this is actually supervised learning because sub-dictionaries are learned for all classes separately.

NMF has been used as a sub-dictionary learning method by MSRC. However, it is unclear if semi-NMF has been used by MSRC in the case of mixed-sign data. Also, if any NMF is used as dictionary learning approach, it is not suitable to pursue the l_1LS sparse coding. This is because a new sample is supposed to be a non-negative linear combination of the dictionary atoms obtained by a NMF, but the coefficient vector obtained by l_1 regularization may contain negative numbers. Thus it makes more sense if NNLS sparse coding is used after the NMF based sub-dictionary learning.

Generally speaking, the sparse coding model must match the dictionary learning model in an implementation of the sub-dictionary learning principle. In this section, we give such an implementation based on non-negative dictionary learning and non-negative sparse coding. This method is called *meta-sample based NNLS* (MNNLS) classification approach, and is described in Algorithm 2.7. A *meta-sample* is defined as a column of the basis matrices obtained via (sub-)dictionary learning. In the algorithm, the meta-samples obtained by NMF from the i-th class are labeled to belong to this class. NMF can be solved by the generic alternating framework described in Section 6.1 or the versatile sparse matrix factorization framework which is discussed in Section 7.

8.1 Computational Experiments

We examined the performance of MNNLS on Dawany's data set [33] which has 5456 microarray gene expression samples distributed in 26 classes (see Section 5.2). We compared MNNLS with NNLS (sparse coding method), NMF (unsupervised dictionary learning), MSRC-SVD, and MSRC-NMF (proposed in [48]). We conducted 10-fold cross-validation to split the data into training and test sets. All the methods are trained on the same training sets. The best mean prediction accuracies, computing time, and corresponding parameters are given in Table 2.6.

Algorithm 2.7 *MNNLS Classification Approach*

Input: $\boldsymbol{D}_{m\times n}$: training data, $\boldsymbol{c}$: class labels of n training samples, $\boldsymbol{B}_{m\times p}$: new data, $\boldsymbol{k}$: size of sub-dictionaries

Output: $\boldsymbol{p}$: the predicted class labels of the p new samples

Training Step:

1. Normalize each training sample to have unit l_2 norm;
2. For any class, learn the sub-dictionary through NMF ($\boldsymbol{D}_{i+} \approx \boldsymbol{A}_{i+}\boldsymbol{Y}_{i+}$ where $\boldsymbol{D}_i$ is the training samples of the i-th class in input space and $\boldsymbol{Y}_{i+}$ is their representation in feature space) if the data are non-negative, or semi-NMF ($\boldsymbol{D}_i \approx \boldsymbol{A}_i\boldsymbol{Y}_{i+}$) if the data are of mixed signs;
3. Concatenate the sub-dictionaries into a holistic dictionary via $\boldsymbol{A} = [\boldsymbol{A}_1, \cdots, \boldsymbol{A}_C]$;

Prediction Step:

1. Normalize each new sample to have unit l_2 norm;
2. Solve the NNLS $\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{B} - \boldsymbol{A}\boldsymbol{X}\|_F^2$ s.t. $\boldsymbol{X} \geq 0$;
3. For each column of $\boldsymbol{X}$, apply nearest-subspace rule to predict the corresponding class label;

Table 2.6: The performance of MNNLS.

Method	Accuracy	Time (seconds)	Parameter
NNLS	0.9762	998	linear kernel
MNNLS	0.9622	618	$k_i = 8$
MSRC-SVD	0.9617	8944	$k_i = 8$, $\lambda = 2^{-10}$
MSRC-NMF	0.7654	341	$k_i = 8$, $\lambda = 2^{-3}$
NMF	0.1391	2.2730e+005	$k = 26$

From the result, first of all, we can see that MNNLS has similar accuracy with the sparse coding method NNLS. Meanwhile, MNNLS took less time than NNLS. Second, the accuracy of MSRC-NMF is very poor, compared to that of MNNLS. This corroborates that the sparse coding method, in a sub-dictionary learning, needs to match with the sub-dictionary learning method carefully. Third, the unsupervised method NMF obtained much worse accuracy, and spent much more time than MNNLS. This confirms that unsupervised dictionary learning may not capture the key patterns of complicated data.

9 Local NNLS Method [12]

In this section, we propose two variants of the NNLS method to classify high dimensional microarray data. Our methods combine the advantages of NMF, local learning, transductive learning, and ensemble learning.

9.1 Related Work and Insights

In this section, we review the concepts of sample selection, local learning, transductive learning, as well as ensemble learning. We also give further insights about them which contribute to our idea presented in Section 9.2.

Sample Selection and Local Learning

Learning on the whole training data might be too complicated due to the unknown distribution of data and the presence of noise or outliers. Therefore *sample selection* has been studied to select a meaningful portion from the whole sample set. Given labeled training data, inductive learning aims to learn a "ubiquitous" model from these training data and make prediction on future unknown samples. Readers are refereed to [50] for a detailed discussion as well as its interaction with feature selection. In an inductive learning, the idea of sample selection is that some samples are more important for (or, relevant to) the classification problem at hand than other samples. For example, *Support vector machine* (SVM) has been utilized to keep only the support vectors (that is, the hard samples) and then select the features given only these hard samples [51]. Alternatively, sample weighting scheme is used in ensemble learning [52] for selecting important samples. Unsupervised learning can also be performed to explore portions of the sample space. For example, *kd-tree* has been used to cluster the training data, such that feature selection is subsequently applied on subset of samples from the resulting clusters [53].

Similar to sample selection, *local learning* [54, 55] provides a much clearer confidence of learning on parts of data. In addition to the reason for sample selection, local learning is a divide-and-conquer scheme when the size of the data is too large to compute a model from. Discrete or smooth kernels are often used to select local samples. Discrete kernels select k samples within the kernel width. Smooth kernels, for instance *radial basis function* (RBF), give more weights to the neighbors in the vicinity of a sample. In fact, *k-nearest neighbors* (k-NN) and *decision tree* are the first-generation of local learning methods [56], and SVM using a RBF kernel is of the second generation of local learning approaches. In [54], a linear

[12] This section is based on our publication [49].

classifier is trained using only k nearest neighbors. *Local SVM* (LSVM) combines both local and transductive learning ideas [57] in which the similarity between an unknown sample and the training samples are computed by using RBF, so that the training phase pays more attention to samples closer to the unknown sample.

Many local learning methods have been devised following its introduction in [54]; see for example [58], [59], and [60]. The partitioning scheme is a crucial step in local learning, in which a recursive tree or kernel is used to cluster the samples. Generally (but not strictly) speaking, if two data points are mutually closest to each other then they are expected to fall into the same cluster. *Profile SVM* (PSVM) [57] uses a supervised grouping method to split the data such that each subgroup equally includes data from both classes before training a linear SVM. Any clustering method can be used in the sample partitioning phase.

Transductive Learning

Local learners such as the k-NN based algorithm [54], LSVM [57], and PSVM [57], are *transductive* methods since unlabeled samples are involved in the partitioning phase before learning a model in each local region. Taking the k-NN based algorithm for example, since each unlabeled but unknown sample is used to select a subset of the training set, the subsequent linear classifiers will makes use of the information contained in the unlabeled samples in order to learn a model. Given a set of labeled data $\boldsymbol{X}$ and a set of unlabeled data $\boldsymbol{S}$, transductive learning learns over the union of $\boldsymbol{X}$ and $\boldsymbol{S}$ in order to predict the correct class labels of the unlabeled samples $\boldsymbol{S}$ [61]. In inductive learning, supervised algorithm is performed only on the labeled data $\boldsymbol{X}$ in order to predicts the class labels of $\boldsymbol{S}$ and any unknown future sample. Transductive learning uses both labeled data $\boldsymbol{X}$ and unlabeled data $\boldsymbol{S}$, and it is also called *semi-supervised learning* [62, 63] since it makes use of both supervised and unsupervised methods.

The advantage of transductive learning is that unlabeled samples contribute to the classification, therefore more information is utilized. This is beneficial for increasing the prediction accuracy, particularly when there are few labeled training data and many unlabeled data available. The objective of transductive learning is different from that of inductive learning. Transductive learning aims to only classify the unlabeled data $\boldsymbol{S}$ and does not predict unlabeled samples not included in $\boldsymbol{S}$. Inductive learning learns a general model from $\boldsymbol{X}$ which is then used to predict the classes of $\boldsymbol{S}$ and any future unknown sample. One shortcoming of transductive learning is that it is time-consuming since it is performed again for every subset of never before seen unlabeled sample; unless an online update method can be devised to update the learned model quickly. Since inductive learning obtains a general

model on the labeled training data, it is thus convenient for predicting any unknown sample. On the other hand, if the task is only to predict the available unlabeled samples once, and if the mixture of labeled and unlabeled data can be clustered well, transductive learning should be better than inductive learning. However, another drawback of transductive learning is that the incorrect prediction of the unlabeled data or ill clustering may propagate and then amplify mistakes in the subsequent prediction phase.

We can relax the single-class constraint in transductive learning, by applying state-of-the-art clustering algorithms, and then allow each resulting cluster to contain samples from more than one classes. This is because it is possible that the distributions of some classes may overlap while still being separable. However, clusters with single class label are still the most welcome. This relaxation can be viewed as a median between the partitioning and the agglomerative approaches. Since a cluster may have more than one unique class label, a classifier must be used in each such cluster.

Ensemble Learning

Local learning is also often combined with *ensemble learning* [64] which is composed of a committee of classifiers and a voting scheme for deciding the final prediction. In [65], each classifier is trained in a random subspace, and the class label of an unknown sample is decided by the weighted vote of many classifiers. As mentioned above, local learning is a divide-and-conquer scheme, and essentially ensemble learning often provides a necessary summary after local learning. *Mixtures of local experts* (MLE) proposed in [66] applies some neural networks, as local experts, each learning on different subregion of the learning space, and then a gating network decides which expert to switch to given a unknown sample. This gating network actually implicitly plays the role of a crisp kernel. Since competitive learning is used in MLE, each expert is actually expected to learn on a subgroup of the complete training data.

9.2 Local NNLS Classifier

Interesting but complicated patterns may be hidden within microarray data. Local learners may approximate this complexity by learning in local regions. In order to emphasize the advantages of local learning, we propose a generalized local version of NNLS which is described in Algorithm 2.8. Please note that, the NNLS classifier given in Section 3 can be viewed a special case of this local approach, if we assume the cluster number is one. Given $\boldsymbol{X}$ containing labeled training samples in columns, the classification task is to categorize the unknown unlabeled samples $\boldsymbol{S}$. Our idea of solving this task is the following. First,

the union of $\boldsymbol{X}$ and $\boldsymbol{S}$ is clustered by NMF (if it is non-negative) or semi-NMF (if it is of mixed signs). Then, for each cluster, if there is no unlabeled sample in this cluster, simply skip this iteration; if this cluster does not contain any training sample, a NNLS classifier is applied over the whole training set $\boldsymbol{X}$ to classify the unlabeled samples in this cluster; if there are both training samples and unlabeled samples in this cluster, then a NNLS classifier learns on these training samples and predict the classes of these unlabeled samples. This approach is also transductive as unlabeled samples are involved in the partitioning phase. This method is named *local NNLS* (LNNLS) classification approach. We denote it as LNNLS-MAX, LNNLS-KNN, and LNNLS-NS for MAX (or NN), K-NN, and NS rules, respectively.

Profile SVM (PSVM) [57] is a local SVM algorithm. The main differences between this approach and PSVM are that i) PSVM has to conduct a supervised partitioning as the learning of a linear SVM in a subgroup require (balanced) samples from both classes, while LNNLS uses unsupervised clustering which neither deteriorate the natural structure of the data, nor require classifiers for some homogeneous clusters (if all the training samples in a cluster are from the same class, then the cluster is *homogeneous*); and ii) PSVM is a binary approach, and hence, computationally costly in the case of multiple classes, whereas LNNLS is appropriate for multi-class data.

Due to the relatively small sample size of microarray data, some readers might be concerned that learning in local regions may worsen the learning performance since the size of a cluster is smaller. We address this concern with the following three points. First, the optimization of NMF and NNLS needs only the inner products of the samples rather than the original high dimensional vectors, and thus, the classification is in fact dimension-free. Replacing the inner products with kernel matrices, we can obtain kernel NMF and kernel NNLS, though we have only investigated linear kernel so far. Second, our LNNLS method does not simply cluster and classify. Applying a classifier in a non-homogeneous cluster (i.e., a cluster containing labeled samples from different classes) is the final step of prediction. For homogeneous clusters, that is when all labeled samples are from the same class c, we can directly assign the class c as the label of the unlabeled samples contained in such clusters. For clusters in which all samples are unlabeled, all training samples will be required for classifying the unlabeled samples. Third, whether the sample size is sufficient or not is a statistical issue, it also depends on the biological and experimental complexity (please see [67] and [68] and references therein). Despite their small sample size, microarray data may still contain enough information needed for discovering the hidden patterns with a statistical learning model. In Section 9.3 we will empirically show that the minimum

number of training samples required by NNLS to obtain a significant accuracy is generally very small on microarray data.

Algorithm 2.8 *LNNLS Classifier*

Input: $\boldsymbol{X}_{m\times n}$: training set with m features and n samples
$\boldsymbol{c}$: class labels of the training samples
$\boldsymbol{S}_{m\times p}$: p unknown samples without labels
Clust_Method: clustering method; *NMF*, *semi-NMF*
Output: $\boldsymbol{p}$: predicted class labels of the p unknown samples

$[\mathbf{X}_i, \mathbf{S}_i]_{i=1}^{i=k} \leftarrow$ *Clustering*$([\boldsymbol{X}, \boldsymbol{S}]$, *Clust_Method*$)$;
$\{\mathbf{X}_i$: *set of labeled samples in i-th cluster*$\}$
$\{\mathbf{S}_i$: *set of unlabeled samples in i-th cluster*$\}$

for $i \leftarrow 1$ **to** k **do**
  if $\boldsymbol{S}_i = \emptyset$ **then**
    Continue;
  end if

  if $\boldsymbol{X}_i = \emptyset$ **then**
    $\boldsymbol{p}_i \leftarrow NNLS(\boldsymbol{X}, \boldsymbol{c}, \boldsymbol{S}_i)$;
      $\{\boldsymbol{p}_i$: *predicted labels of samples in* $\boldsymbol{S}_i\}$
    Continue;
  end if

  if *Homogeneous*$(\boldsymbol{X}_i)$ **then**
    $\boldsymbol{p}_i \leftarrow \boldsymbol{c}_i$; $\{\boldsymbol{c}_i$ *is the unique class label appearing in* $\boldsymbol{X}_i\}$
    Continue;
  end if
  $\boldsymbol{p}_i \leftarrow NNLS(\boldsymbol{X}_i, \boldsymbol{c}_i, \boldsymbol{S}_i)$;
end for

return $\boldsymbol{p}$.

9.3 Repetitive LNNLS Classifier

The performance of the LNNLS classifier depends on the clustering algorithm (NMF or semi-NMF). Due to the non-convexity of NMF and to the random initializations, different executions of NMF may result in different factors, and therefore we may obtain different clustering results. We thus propose the *repetitive LNNLS* (RLNNLS) classification approach, in Algorithm 2.9, in which we perform LNNLS learning on the data *maxR* times.

The final class labels of the unlabeled samples are then decided through a voting process on the decisions returned by the *maxR* LNNLS classifiers. The *majority rule* is used as our voting method. Thus, this technique falls in the domain of ensemble learning [64]. From now on, RLNNLS coupled with MAX, K-NN, and NS rules are denoted as *RLNNLS-MAX*, *RLNNLS-KNN*, and *RLNNLS-NS*, respectively.

Algorithm 2.9 *RLNNLS Classifier*

Input: $\boldsymbol{X}_{m\times n}$: training set with m features and n samples
$\boldsymbol{c}$: class labels of the training samples
$\boldsymbol{S}_{m\times p}$: p unknown samples without labels
Output: $\boldsymbol{p}$: predicted class labels of the p unknown samples

for $r \leftarrow 1$ **to** *maxR* **do**
$\mathbf{p}_r \leftarrow LNNLS(\boldsymbol{X}, \boldsymbol{c}, \boldsymbol{S})$; $\{\mathbf{p}_r$: *predicted labels at the r-th iteration*$\}$
end for

return $\boldsymbol{p} \leftarrow Majority\_Vote(\mathbf{p}_1, \ldots, \mathbf{p}_{maxR})$.

Experiment

In order to investigate the classification performance of our family of NNLS classifiers for high dimensional biological data, we ran it on 14 microarray gene expression data sets (including 8 two-class and 6 multi-class data sets) and 3 two-class array-based comparative genomic hybridization (aCGH) data sets. The aCGH technique is used to identify DNA copy numbers. These data are summarized in Table 2.7. The aCGH data sets are in the last block of the table. The numbers of features vary from 2000 to 24481 among these 17 data sets. The numbers of samples vary from 34 to 248. The gene expression intensities are naturally non-negative. However, due to preprocessing when producing the data, some data have negative components. 5 out of these 14 data sets, are non-negative. The aCGH data are presented as $\log_2$-ratio, therefore have both positive and negative values. In this study, for non-negative data, NMF is used to cluster the data; otherwise, semi-NMF is employed to do the same task. The optimal number of clusters was obtained by linear search, though a model selection method proposed in [75] is widely used. This is because we observed that the optimal number of clusters suggested by that model selection method does not always result in the highest classification accuracy. We used linear kernel in our NNLS family. We compared our NNLS family with three classical local learners, namely, k-NN, SVM with RBF kernel, and LSVM. We implemented all these methods in MATLAB. The

Table 2.7: High dimensional microarray data sets.

Data	+/-	#Classes	#Features	#Samples
Adenoma[69]	±	2	7457	18+18=36
Breast[70]	+	2	24481	44+34=74
Colon[44]	+	2	2000	40+22=62
DLBCL-NIH[71]	±	2	7399	102+138=240
Leukemia[72]	±	2	7129	47+25=72
Lung[73]	±	2	7129	10+86=96
Medulloblastoma[74, 75]	+	2	5893	25+9=34
Prostate[76]	±	2	12600	59+77=136
ALL[77]	±	6	12625	15+27+64+20+43+79=248
ALLAML[72, 75]	+	3	5000	19+8+11=38
Breast5[31]	±	5	13582	39+22+53+31+13=158
CNS[78]	±	5	7129	10+10+10+4+8=42
MLL[79]	±	3	12582	24+20+28=72
SRBCT[80]	+	4	2308	23+8+12+20=63
Bladder[81]	±	2	2143	32+16=48
BreastBerkeley[82]	±	2	2149	72+69=141
Melanoma[83]	±	2	3649	35+43=78

recently proposed *sparse partial least squares discriminant analysis* (SPLSDA) and *sparse generalized PLS* (SGPLS) [84] were also compared with our methods. We used the SPLS package (version 2.1-1) in R which is available from [84]. Parameters of all methods were selected by linear or grid search. Four-fold cross-validation was used to split a whole data set into labeled training set and unlabeled test set. We defined the *accuracy* of a given classifier as the ratio of the number of correctly predicted test samples to the total number of test samples. For each data set, four-fold cross-validation was rerun 20 times, and the averages and standard deviations over the 20 runs were computed.

Figure 2.6 shows the results on six data sets: Breast, Colon, Prostate, ALL, Breast5, and BreastBerkeley. SGPLS fails on the ALL data set which has a very large number of features. LSVM gives better performance compared with SVM on Breast, ALL, and BreastBerkeley, and gives very poor performance on Colon and Prostate. From all six groups of results, it can be seen that the medium performance was obtained by k-NN. It can also be seen that our NNLS classifiers generally perform very well. We can also see that NNLS-NS has better accuracies than NNLS-MAX on Prostate and Breast5. We can observe that LNNLS-NS

obtained better results than NNLS-NS. Furthermore, the RLNNLS-NS classifier has better accuracies than LNNLS-NS on Breast, Prostate, and ALL; meanwhile comparable results were obtained by both of them over the remaining three data sets.

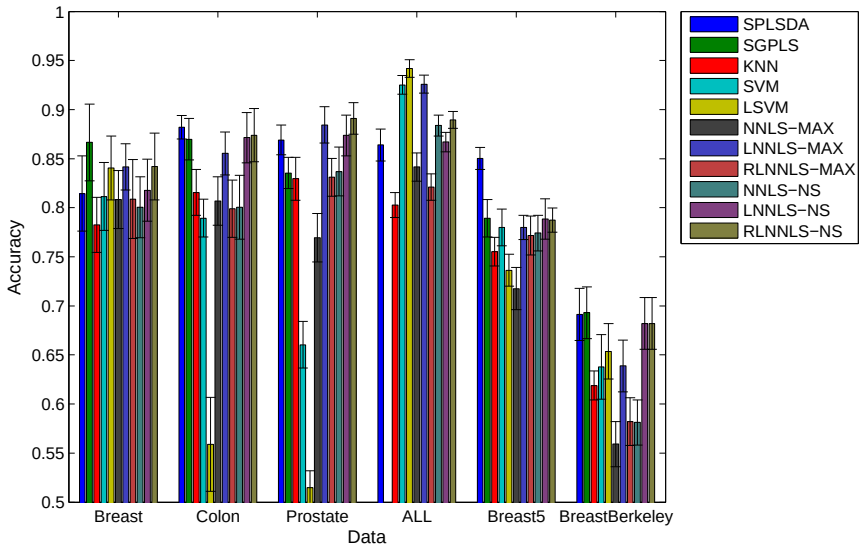


Figure 2.6: Classification performance on six data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

Statistical Comparison

We used the Friedman test with the post-hoc Nemenyi test to further statistically compare all the classifiers over their mean accuracies on the 17 data sets. The Friedman test is a non-parametric approach which compares multiple classifiers on multiple data sets [85]. It has been recommended by [85], since it is simple, safe, and robust. In the Friedman test, the classifiers are ranked for each data set. The classifier obtained the best performance has rank 1, and the second best classifier gets rank 2, and so on. The rank of ties is their rank average. For example the 3-rd and 4-th classifiers have the same performance on a data set, then both of them get rank 3.5. The average rank of each classifier are then obtained over all data sets. The statistic of the test is a function of the average ranks of all classifiers:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2}, \tag{2.105}$$

where N is the number data sets, k is the number of classifiers, and χ_F^2 is the Friedman statistic, and is defined as

$$\chi_F^2 = \frac{12N}{k(k+1)}\Big(\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\Big), \tag{2.106}$$

where R_j is the average rank of the j-th classifier. Statistic F_F follows F-distribution with degrees of freedom $(k-1, (k-1)(N-1))$. The null hypothesis of the Friedman test is that all classifiers are equivalent in term of error rate. Once it is rejected, the Nemenyi test is then used to find the difference among the classifiers. In the Nemenyi test, the *crucial difference* (CD) is computed as

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}, \tag{2.107}$$

where α is the significance level, and q_α is the upper crucial value (upper quantile) of the Studentized range distribution (with infinite degree of freedom) divided by $\sqrt{2}$. During the Nemenyi test, if the distance between the average ranks of a pair of classifiers is larger than the CD, then we say that they are significantly different. The significance level was set to $\alpha = 0.05$ in our experiment. The null hypothesis was rejected in our test, and thus the post-hoc Nemenyi test was conducted. The graphical presentation of the Nemenyi test can be found in Figure 2.7 . According to the test results, we have the following interpretations which are consistent with our above observations from Figure 2.6.

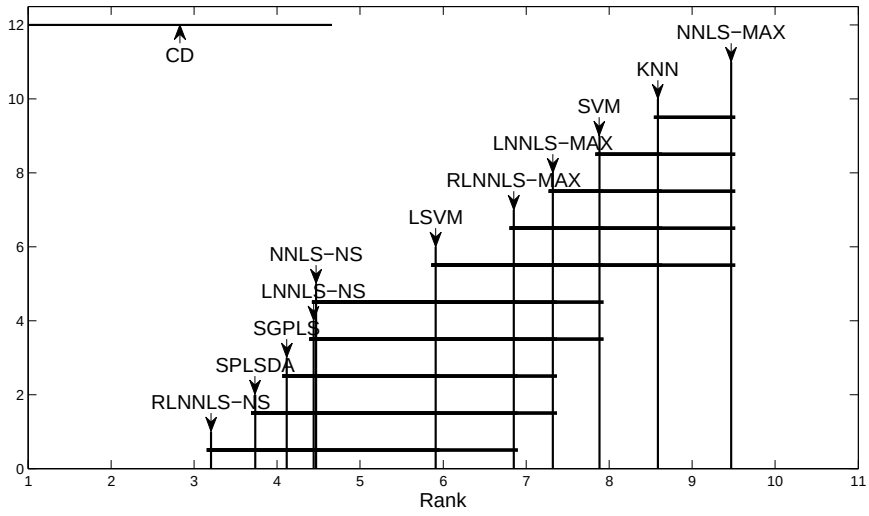


Figure 2.7: Crucial difference diagram of the Nemenyi test ($\alpha = 0.05$).

NNLS using NS rule is significantly better than the one using MAX rule. NNLS-NS is significantly better than k-NN. There is a large difference between the average ranks of NNLS-NS and SVM, though the difference is not significant at the current significance level. If we increase α slightly, NNLS-NS and SVM will be significantly different. The average rank of LNNLS is not worse than NNLS. RLNNLS-NS obtained the best average rank. It is significantly different from LNNLS-MAX, SVM, k-NN, and NNLS-MAX. Finally, the average ranks of SPLSDA and SGPLS are between those of RLNNLS-NS and LNNLS, and current test does not find significant difference among them.

Computing Time

The averaged elapsed execution time in seconds of each method was also recorded as a measure of performance. All experiments were performed on an Intel machine (Core TM i7, 2.93 GHz, CPU with 8 GB RAM, with 64-bit Windows 7 Professional operation system). All methods, except the SPLS methods, were implemented in the language MATLAB, 64-bit version 7.11. The SPLS methods was implemented in the language R, 64-bit version 2.12. The computing time of LNNLS and RLNNLS does not include the time of clustering, because clustering can be done only once and then the results can be saved in memory or hard drive. During the iterations of cross-validation, the same clustering results can be reused through simply changing the roles (training or test) of the samples. In real-world applications, to predict the class labels of new but unlabeled samples, the computing time may increase since clustering must be redone. *Online* updating methods can be investigated to reduce the computing time of NMF clustering; see [86] for an example of an online NMF. Figure 2.8 shows the computing time of the methods on the six data sets; $\log_2 times$ are given for a better visual comparison.

First, it can be clearly seen that our NNLS approaches are much more efficient than the PLS methods over large or multi-class data. For instance on ALL, the computing time of SGPLS is as 10 and 7 times as NNLS-NS and RLNNLS-NS, respectively. Second, our inductive and local NNLS methods are also consistently faster than local SVM. Local SVM becomes computationally costly when the number of classes is large. For example, over Breast5 and ALL, it performs slower than our RLNNLS methods. One of the reason for the efficiency of our methods, compared to PLS and LSVM methods, is because our techniques are naturally multi-class methods, while the supervised PLS and LSVM are binary models which have to use one-against-all, one-against-one, or other strategies for multi-class task. Therefore their computing time is significantly increased in the multi-class case.

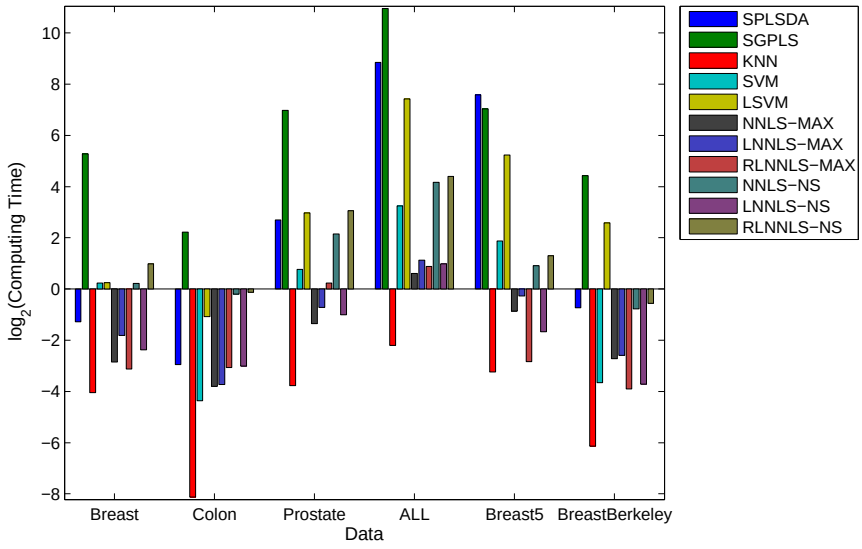


Figure 2.8: Computing time on six data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

Estimation of Data-Size Requirement

We estimated the minimum data size (that is, the number of training samples) required to obtain significant accuracy, in order to explore the practical reason for the high performance of our approaches. We implemented the permutation-test-based method proposed in [67]. We set the significance level to $\alpha = 0.05$ in these experiments. The minimum data size of NNLS-NS and SVM for each data set is listed in Table 2.8. In the experiments above, all classifiers obtained poor performance on DLBCL-NIH. Through investigating the data-size requirements, we found that the sample size provided in this data set was not sufficient to obtain a significant accuracy. In Table 2.8, we can see that NNLS-NS generally needs fewer training samples in order to achieve significant accuracy. This also is a possible reason of why LNNLS methods obtained good accuracies, in practice, as they perform prediction via clustering.

10 Conclusions

In this chapter, we proposed the Bayesian sparse coding and dictionary learning models. We intensively investigated three sparse coding models: the l_1-regularized model, the non-negative model, and l_1-regularized non-negative model. We revealed that these models can

Table 2.8: Minimum data size required to obtain significant accuracy ($\alpha = 0.05$). NNLS-NS method only requires a very small number of training samples in order to obtain significant accuracy.

Data	NNLS-NS	SVM
Adenoma	6	2
Breast	10	12
Colon	8	12
DLBCL-NIH	-	-
Leukemia	6	12
Lung	15	11
Medulloblastoma	12	2
Prostate	9	11
ALL	6	10
ALLAML	5	10
Breast5	5	11
CNS	5	14
MLL	4	18
SRBCT	4	10
Bladder	10	8
BreastBerkeley	20	11
Melanoma	6	4

be kernelized. We reviewed main sparse coding optimization methods, and proposed our own active-set method and decomposition method. We proposed a generic network of kernel dictionary learning. We then proposed a more general framework, named versatile sparse matrix factorization. The classification, feature extraction, feature selection techniques based on sparse representation for high-dimensional biological data were investigated. We proposed two novel classification techniques – the sub-dictionary learning and local sparse coding for complicated biological data.

There are some interesting future researches. We just mentioned a few in the following. First, a novel supervised dictionary learning can be devised by combining Bayesian dictionary learning and Bayesian regression. Second, it would be interesting to investigate supervised kernel dictionary learning. Third, optimization methods based on Markov chain Monte Carlo sampling can be implemented for the sparse representation models.

Publications

1. We first discussed the sparse representation from a Bayesian viewpoint in [*Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," BMC Systems Biology, 2013, in press.*].

2. We proposed the non-negative least squares based sparse coding model and investigate their classification performance in [*Y. Li and A. Ngom, "Classification approach based on non-negative least squares," Neurocomputing, vol. 118, pp. 41-57, 2013.*].

3. We proposed the weighted K-nearest neighbor rule for predicting class labels based on the sparse coefficient obtained by a sparse coding model in [*Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," BMC Systems Biology, 2013, in press.*].

4. Kernel sparse coding models were discussed in [*Y. Li and A. Ngom, "Classification approach based on non-negative least squares," Neurocomputing, vol. 118, pp. 41-57, 2013.*] and [*Y. Li and A. Ngom, "Fast sparse representation approaches for the classification of high-dimensional biological data," IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Philadelphia, Oct. 2012, pp. 306-311.*].

5. We preliminarily proposed the efficient active-set methods in [*Y. Li and A. Ngom, "Fast kernel sparse representation approaches for classification," IEEE International Conference on Data Mining (ICDM), Brussels, Belgium, Dec. 2012, pp. 966-971.*], and decomposition methods in [*Y. Li, R.J. Caron, and A. Ngom, "A decomposition method for large-scale sparse coding in representation learning," IEEE World Congress in Computational Intelligence (WCCI), July 2014, to be submitted.*] for learning the parameter of the sparse coding model.

6. We proposed two unified frameworks for kernel dictionary learning in [*Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," BMC Systems Biology, 2013.*] and [*Y. Li and A. Ngom, "Versatile sparse matrix factorization and its applications in high-dimensional biological data analysis," IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB), Nice, June, 2013, LNBI 7986, pp. 91-101.*].

7. We first proposed the kernel non-negative matrix factorization for the dimensionality reduction of microarray data in [*Y. Li and A. Ngom, "A new kernel non-negative*

matrix factorization and its application in microarray data analysis," IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), San Diego, CA, May 2012, pp. 371-378.], and then the generic framework of kernel dictionary learning model is presented in [*Y. Li and A. Ngom, "Non-negative matrix and tensor factorization based classification of clinical microarray gene expression data," IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Hong Kong, Dec. 2010, pp.438-443.*], and then the extended version of this paper is published in the special issue [*Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," BMC Systems Biology, 2013, in press.*].

8. The supervised dictionary learning model for multi-class data based on sub-dictionary learning was originally published in [*Y. Li and A. Ngom, "Supervised dictionary learning via non-negative matrix factorization for classification," International Conference on Machine Learning and Applications (ICMLA), Boca Raton, Florida, Dec. 2012, pp. 439-443.*].

9. The local NNLS methods were originally proposed in [*Y. Li and A. Ngom, "Non-negative least squares methods for the classification of high dimensional biological data," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 10, no. 2, pp. 447-456, 2013.*].

10. My thesis study on sparse representation was introduced in [*Y. Li, "Sparse representation for machine learning," 26th Canadian Conference on Artificial Intelligence (AI 2013), Regina, May, 2013, LNAI 7884, pp. 352-357.*].

Chapter 3

The Non-Negative Matrix Factorization and Sparse Representation Toolboxes

1 Introduction

Open-source packages are very important for both researchers in the machine learning and bioinformatics communities. From receiving feedbacks and suggestions from users, the developers can improve their implementations, and add new functionalities. By using these packages, researchers in the communities enjoy the convenience of using implementations of most important methods for a computational theory.

We have already implemented two toolboxes including *non-negative matrix factorization* (NMF) toolbox and *sparse representation* (SR) toolbox. NMF and SR have evolved relatively independently in the machine learning field, though NMF is essentially a non-negative sparse representation model. NMF has been intensively studied in both machine learning and bioinformatics, while the general SR is still not popular in bioinformatics. For this reason, we separate the implementations of NMF and SR. Since NMF is a special case of sparse representation, it is almost unavoidable that there are some overlaps in the both implementations. However, we made efforts to minimize overlapping.

Contributions: In this chapter, we describe our implementations of the sparse representation models which have already been derived in Chapter 2. Our implementations include two toolboxes: NMF toolbox and SR toolbox. The following are our contributions:

1. The NMF algorithms, in our NMF toolbox, are relatively complete and implemented in MATLAB.

2. Our NMF toolbox includes many functionalities for mining biological data, such as clustering, biclustering, feature extraction, feature selection, and classification.

3. The NMF toolbox also provides additional functions for biological data visualization, such as heat-maps and other visualization tools. They are pretty helpful for interpreting some results. Statistical methods are also included for comparing the performances of multiple methods.

4. The SR toolbox consists of all the sparse coding and dictionary learning algorithms discussed Chapter 2.

5. Based on the basic level of the SR algorithms, machine learning methods, including classification and dimensionality reduction, are implemented in our SR toolbox.

In the following, we first introduce our NMF toolbox in Section 2, and then the SR toolbox in Section 3. Due to the importance of NMF, we will briefly describe important variants of NMF in the section 2.2.

2 The Non-Negative Matrix Factorization Toolbox [1]

2.1 Background

Non-negative matrix factorization (NMF) is a matrix decomposition approach which decomposes a non-negative matrix into two low-rank non-negative matrices [12]. It has been successfully applied in the mining of biological data.

For example, the authors of [75] and [38] used NMF as a clustering method in order to discover the metagenes (i.e., groups of similarly behaving genes) and interesting molecular patterns. The authors of [87] applied *non-smooth NMF* (NS-NMF) for the biclustering of gene expression data. *Least-squares NMF* (LS-NMF) was proposed to take into account the uncertainty of the information present in gene expression data [88]. We have [89] proposed the kernel NMF for reducing the dimensions of gene expression data.

Many authors indeed provide their respective NMF implementations along with their publications so that the interested community can use them to perform the same data mining tasks respectively discussed in those publications. However, there exist at least

[1] This section is based on our publication [11].

three issues that prevent NMF methods from being used by the much larger community of researchers and practitioners in the data mining, biological, health, medical, and bioinformatics areas. First, these NMF software packages are implemented in diverse programming languages, such as R, MATLAB, C++, and Java, and usually only one optimization algorithm is provided in their packages. It is inconvenient for many researchers who want to choose a suitable NMF method or mining task for their data, among the many different implementations, which are realized in different languages with different mining tasks, control parameters, or criteria. Second, some papers only provide NMF optimization algorithms at a basic level rather than a data mining implementation at a higher level. For instance, it becomes hard for a biologist to fully investigate and understand his/her data when performing clustering or biclustering of his data and then visualize the results; because it should not be necessary for him/her to implement these three data mining methods based on a basic NMF. Third, the existing NMF implementations are application-specific, and thus, there exists no systematic NMF package for performing data mining tasks on biological data.

There currently exist NMF toolboxes, however, none of them addresses the above three issues altogether. *NMFLAB* [90] is a MATLAB toolbox for signal and image processing which provides a user-friendly interface to load and process input data, and then save the results. It includes a variety of optimization algorithms such as multiplicative rules, exponentiated gradient, projected gradient, conjugate gradient, and quasi-Newton methods. It also provide methods for visualizing the data signals and their components, but does not provide any data mining functionality. Other NMF approaches such as semi-NMF and kernel NMF are not implemented within this package. *NMF:DTU Toolbox* [91] is a MATLAB toolbox with no data mining functionalities. It includes only five NMF optimization algorithms, such as multiplicative rules, projected gradient, probabilistic NMF, alternating least squares, and alternating least squares with *optimal brain surgery* (OBS) method. *NMFN: Non-negative Matrix Factorization* [92] is a R package similar to *NMF:DTU* but with few more algorithms. *NMF: Algorithms and framework for Nonnegative Matrix Factorization* [93] is another R package which implements several algorithms and allows parallel computations but no data mining functionalities. *Text to Matrix Generator* (TMG) is a MATLAB toolbox for text mining only. The authors of [94] provides a NMF plug-in for BRB-ArrayTools. This plug-in only implements the standard NMF and semi-NMF and for clustering gene expression profiles only. *Coordinated Gene Activity in Pattern Sets* (CoGAPS) [95] is a new package implemented in C++ with R interface. In this package, the *Bayesian decomposition* (BD) algorithm is implemented and used in place of the NMF method for factorizing a matrix. Statistical methods are also provided for the inference of

biological processes. CoGAPS can give more precise results than NMF methods [1]. However, CoGAPS uses a *Markov chain Monte Carlo* (MCMC) scheme for estimating the BD model parameters; which is slower than the NMFs optimization algorithms implemented with the block-coordinate gradient descent scheme.

In order to address the lack of data mining functionalities and the generalities of current NMF toolboxes, we propose a general NMF toolbox in MATLAB which is implemented in two levels. The basic level is composed of the different variants of NMF, and the top level consists of the diverse data mining methods for biological data. The source code of this toolbox can be downloaded at `https://sites.google.com/site/nmftool` and `http://cs.uwindsor.ca/~li11112c/nmf`.

2.2 Implementation

As mentioned above, this toolbox is implemented at two levels. The fundamental level is composed of several NMF variants and the advanced level includes many data mining approaches based on the fundamental level. The critical issues in implementing these NMF variants are addressed in this section. Table 3.1 summarizes all the NMF algorithms implemented in our toolbox. Users (researchers, students, and practitioners) should use the command `help nmfrule`, for example, in the command line, for help on how to select a given function and set its parameters.

Standard-NMF

The *standard-NMF* decomposes a non-negative matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$ into two non-negative factors $\boldsymbol{A} \in \mathbb{R}^{m\times k}$ and $\boldsymbol{Y} \in \mathbb{R}^{k\times n}$ (where $k < \min\{m, n\}$), that is

$$\boldsymbol{X}_+ = \boldsymbol{A}_+\boldsymbol{Y}_+ + \boldsymbol{E}, \tag{3.1}$$

where, $\boldsymbol{E}$ is the error (or residual) and $\boldsymbol{M}_+$ indicates the matrix $\boldsymbol{M}$ is non-negative. Its optimization in the Euclidean space is formulated as

$$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_F^2, \text{s.t. } , \boldsymbol{A}, \boldsymbol{Y} \geq 0. \tag{3.2}$$

Statistically speaking, this formulation is obtained from the log-likelihood function under the assumption of a Gaussian error. If multivariate data points are arranged in the columns of $\boldsymbol{X}$, then $\boldsymbol{A}$ is called the *basis matrix* and $\boldsymbol{Y}$ is called the *coefficient matrix*; each column of $\boldsymbol{A}$ is thus a *basis vector*. The interpretation is that each data point is a (sparse) non-negative

Table 3.1: Algorithms of NMF variants.

Function	Description
nmfrule	The standard NMF optimized by gradient-descent-based multiplicative rules.
nmfnnls	The standard NMF optimized by NNLS active-set algorithm.
seminmfrule	Semi-NMF optimized by multiplicative rules.
seminmfnnls	Semi-NMF optimized by NNLS.
sparsenmfnnls	Sparse-NMF optimized by NNLS.
sparsenmfNNQP	Sparse-NMF optimized by NNQP.
sparseseminmfnnls	Sparse semi-NMF optimized by NNLS.
kernelnmfdecom	Kernel NMF through decomposing the kernel matrix of input data.
kernelseminmfrule	Kernel semi-NMF optimized by multiplicative rule.
kernelseminmfnnls	Kernel semi-NMF optimized by NNLS.
kernelsparseseminmfnnls	Kernel sparse semi-NMF optimized by NNLS.
kernelSparseNMFNNQP	Kernel sparse semi-NMF optimized by NNQP.
convexnmfrule	Convex-NMF optimized by multiplicative rules.
kernelconvexnmf	Kernel convex-NMF optimized by multiplicative rules.
orthnmfrule	Orth-NMF optimized by multiplicative rules.
wnmfrule	Weighted-NMF optimized by multiplicative rules.
sparsenmf2rule	Sparse-NMF on both factors optimized by multiplicative rules.
sparsenmf2nnqp	Sparse-NMF on both factors optimized by NNQP.
vsmf	Versatile sparse matrix factorization optimized by NNQP and l_1QP.
nmf	The omnibus of the above algorithms.
computeKernelMatrix	Compute the kernel matrix $k(\boldsymbol{A}, \boldsymbol{B})$ given a kernel function.

linear combination of the basis vectors. It is well-known that the objective is non-convex, and thus, *block-coordinate descent* is the main prescribed optimization technique for such a problem. Multiplicative update rules were introduced in [40] for solving the optimization problem (3.2). Though simple to implement, this algorithm is not guaranteed to converge to a stationary point [41]. Essentially the optimizations above, with respect to $\boldsymbol{A}$ and $\boldsymbol{Y}$, are *non-negative least squares* (NNLS). Therefore we implemented the alternating NNLS algorithm proposed in [41]. It can be proven that this algorithm converges to a stationary point [41]. In our toolbox, functions `nmfrule` and `nmfnnls` are the implementations of the two algorithms above.

Semi-NMF

The standard NMF only works for non-negative data; which limits its applications. The authors of [35] have extended it to *semi-NMF* which removes the non-negative constraints

on the data $\boldsymbol{X}$ and basis matrix $\boldsymbol{A}$. It can be expressed in the following equation:

$$\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_F^2, \text{ s.t. } \boldsymbol{Y} \geq 0. \tag{3.3}$$

Semi-NMF can be applied to the matrix of mixed signs, therefore it expands NMF to many fields. However, the gradient-descent-based update rule proposed in [35] is slow to converge (implemented in function `seminmfrule` in our toolbox). Keeping $\boldsymbol{Y}$ fixed, updating $\boldsymbol{A}$ is a least squares problem which has an analytical solution

$$\boldsymbol{A} = \boldsymbol{X}\boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1} = \boldsymbol{X}\boldsymbol{Y}^{\dagger}, \tag{3.4}$$

where $\boldsymbol{Y}^{\dagger} = \boldsymbol{Y}^{\mathrm{T}}(\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}})^{-1}$ is the Moore-Penrose pseudoinverse. Updating $\boldsymbol{Y}$ while fixing $\boldsymbol{A}$ is a NNLS problem essentially as above. Therefore we implemented the fast NNLS based algorithm to optimize semi-NMF in function `seminmfnnls`.

Sparse-NMF

The standard NMF and semi-NMF have the issues of scale-variance and non-unique solutions, which imply that the non-negativity constrained on the least squares is insufficient in some cases. Sparsity is a popular regularization principle in statistical modeling [16], and has already been used in order to reduce the non-uniqueness of solutions and also enhance the interpretability of the NMF results. The *sparse-NMF* proposed in [38] is expressed in the following equation

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X} - \boldsymbol{A}\boldsymbol{Y}\|_F^2 + \frac{\eta}{2}\|\boldsymbol{A}\|_F^2 + \frac{\lambda}{2}\sum_{i=1}^{n}\|\boldsymbol{y}_i\|_1^2 \\ &\text{s.t. } \boldsymbol{A}, \boldsymbol{Y} \geq 0, \end{aligned} \tag{3.5}$$

where, $\boldsymbol{y}_i$ is the i-th column of $\boldsymbol{Y}$, η and λ are pre-specified control parameters. From the Bayesian perspective, this formulation is obtained from the log-posterior probability under the assumptions of Gaussian error, Gaussian-distributed basis vectors, and Laplace-distributed coefficient vectors. Keeping one matrix fixed and updating the other matrix can be formulated as a NNLS problem. In order to improve the interpretability of the basis

vectors and speed up the algorithm, we implemented the following model instead:

$$\begin{aligned} &\min_{\boldsymbol{A},\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \lambda \sum_{i=1}^{n} \|\boldsymbol{y}_i\|_1 \\ &\text{s.t. } \boldsymbol{A},\boldsymbol{Y} \geq 0, \\ &\quad \|\boldsymbol{a}_i\|_2^2 = 1, \quad i = 1, \cdots, k. \end{aligned} \tag{3.6}$$

We optimize this using three alternating steps in each iteration. First, we optimize the following task:

$$\begin{aligned} &\min_{\boldsymbol{Y}} \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \lambda \sum_{i=1}^{n} \|\boldsymbol{y}_i\|_1 \\ &\text{s.t. } \boldsymbol{Y} \geq 0. \end{aligned} \tag{3.7}$$

then, $\boldsymbol{A}$ is updated as follows:

$$\begin{aligned} &\min_{\boldsymbol{A}} \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 \\ &\text{s.t. } \boldsymbol{A} \geq 0. \end{aligned} \tag{3.8}$$

and then, the columns of $\boldsymbol{A}$ are normalized to have unit l_2 norm. The first and second steps can be solved using *non-negative quadratic programming* (NNQP), whose general formulation is

$$\begin{aligned} &\min_{\boldsymbol{Z}} \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{H}\boldsymbol{z}_i + \boldsymbol{g}_i^{\mathrm{T}}\boldsymbol{z}_i + c_i \\ &\text{s.t. } \boldsymbol{Z} \geq 0, \end{aligned} \tag{3.9}$$

where, $\boldsymbol{z}_i$ is the i-th column of the variable matrix $\boldsymbol{Z}$. It is easy to prove that NNLS is a special case of NNQP. For example, Equation (3.7) can be rewritten as

$$\begin{aligned} &\min_{\boldsymbol{Y}} \sum_{i=1}^{n} \frac{1}{2}\boldsymbol{y}_i^{\mathrm{T}}(\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})\boldsymbol{y}_i + (\lambda - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{x}_i)^{\mathrm{T}}\boldsymbol{y}_i + \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_i \\ &\text{s.t. } \boldsymbol{Y} \geq 0. \end{aligned} \tag{3.10}$$

The implementations of the method in [38] and our method are given in functions `sparsenmfnnls` and `sparseNMFNNQP`, respectively. We have also implemented the

sparse semi-NMF in function `sparseseminmfnnls`.

Versatile Sparse Matrix Factorization

When the training data $\boldsymbol{X}$ is of mixed signs, the basis matrix $\boldsymbol{A}$ is not necessarily constrained to be non-negative; this depends on the application or the intentions of the users. However, without non-negativity, $\boldsymbol{A}$ is not sparse any more. In order to obtain a sparse basis matrix $\boldsymbol{A}$ for some analysis, we may use l_1-norm on $\boldsymbol{A}$ to induce sparsity. The drawback of l_1-norm is that correlated variables may not be simultaneously non-zero in the l_1-induced sparse result. This is because l_1-norm is able to produce sparse but non-smooth results. It is known that l_2-norm is able to obtain smooth but non-sparse results. When both norms are used together, then correlated variables can be selected or removed simultaneously [37]. When smoothness is required on $\boldsymbol{Y}$, we may also use l_2-norm on it in some scenarios. We thus generalize the aforementioned NMF models into a versatile form as expressed below

$$\min_{\boldsymbol{A},\boldsymbol{Y}} f(\boldsymbol{A},\boldsymbol{Y}) = \frac{1}{2}\|\boldsymbol{X}-\boldsymbol{A}\boldsymbol{Y}\|_F^2 + \sum_{i=1}^{k}(\frac{\alpha_2}{2}\|\boldsymbol{a}_i\|_2^2 + \alpha_1\|\boldsymbol{a}_i\|_1) + \sum_{i=1}^{n}(\frac{\lambda_2}{2}\|\boldsymbol{y}_i\|_2^2 + \lambda_1\|\boldsymbol{y}_i\|_1) \tag{3.11}$$

$$\text{s.t. } \begin{cases} \boldsymbol{A} \geq 0 & \text{i.e., if } t_1 = 1 \\ \boldsymbol{Y} \geq 0 & \text{i.e., if } t_2 = 1 \end{cases},$$

where, parameters: $\alpha_1 \geq 0$ controls the sparsity of the basis vectors; $\alpha_2 \geq 0$ controls the smoothness and the scale of the basis vectors; $\lambda_1 \geq 0$ controls the sparsity of the coefficient vectors; $\lambda_2 \geq 0$ controls the smoothness of the coefficient vectors; and, parameters t_1 and t_2 are boolean variables (0: false, 1: true) which indicate if non-negativity needs to be enforced on $\boldsymbol{A}$ or $\boldsymbol{Y}$, respectively. We can call this model *versatile sparse matrix factorization* (VSMF). It can be easily seen that the standard NMF, semi-NMF, and the sparse-NMFs are special cases of VSMF.

We devise the following multiplicative update rules for the VSMF model in the case of $t_1 = t_2 = 1$ (implemented in function `sparsenmf2rule`):

$$\begin{cases} \boldsymbol{A} = \boldsymbol{A} * \frac{\boldsymbol{X}\boldsymbol{Y}^{\mathrm{T}}}{\boldsymbol{A}\boldsymbol{Y}\boldsymbol{Y}^{\mathrm{T}}+\alpha_2\boldsymbol{A}+\alpha_1} \\ \boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}}{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{Y}+\lambda_2\boldsymbol{Y}+\lambda_1} \end{cases}, \tag{3.12}$$

where, operations $\boldsymbol{U} * \boldsymbol{V}$ and $\frac{\boldsymbol{U}}{\boldsymbol{V}}$ are the element-wise multiplication and division operators of matrices $\boldsymbol{U}$ and $\boldsymbol{V}$, respectively. Alternatively, we also devise an active-set algorithm for

VSMF (implemented in function `vsmf`). When t_1(or t_2) = 1, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated by NNQP (this case is also implemented in `sparsenmf2nnqp`). When t_1(or t_2) = 0, $\boldsymbol{A}$ (or $\boldsymbol{Y}$) can be updated using 1_1QP.

Kernel-NMF

Two features of a kernel approach are that i) it can represent complex patterns, and ii) the optimization of the model is dimension-free. We now show that NMF can also be kernelized.

The basis matrix is dependent on the dimension of the data, and it is difficult to represent it in a very high (even infinite) dimensional space. We notice that in the NNLS optimization, updating $\boldsymbol{Y}$ in Equation (3.10) needs only the inner products $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X}$, and $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. From Equation (3.4), we obtain $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = (\boldsymbol{Y}^{\dagger})^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{Y}^{\dagger}$, $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{X} = (\boldsymbol{Y}^{\dagger})^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. Therefore, we can see that only the inner product $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ is needed in the optimization of NMF. Hence, we can obtain the kernel version, *kernel-NMF*, by replacing the inner product $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$ with a kernel matrix $K(\boldsymbol{X}, \boldsymbol{X})$. Interested readers can refer to our recent paper [89] for further details. Based on the above derivations, we implemented the kernel semi-NMF using multiplicative update rule (in `kernelseminmfrule`) and NNLS (in `kernelseminmfnnls`). The sparse kernel semi-NMFs are implemented in functions `kernelsparseseminmfnnls` and `kernelSparseNMFNNQP` which are equivalent to each other. The kernel method of decomposing a kernel matrix proposed in [96] is implemented in `kernelnmfdecom`.

Other Variants

The authors of [35] proposed the *Convex-NMF*, in which one column of $\boldsymbol{A}$ is constrained to be a convex combination of data points in $\boldsymbol{X}$. It is formulated as $\boldsymbol{X}_{\pm} = \boldsymbol{X}_{\pm}\boldsymbol{W}_{+}\boldsymbol{Y}_{+} + \boldsymbol{E}$, where $\boldsymbol{M}_{\pm}$ indicates that matrix $\boldsymbol{M}$ is of mixed signs. $\boldsymbol{X}\boldsymbol{W} = \boldsymbol{A}$ and each column of $\boldsymbol{W}$ contains the convex coefficients of all the data points to get the corresponding column of $\boldsymbol{A}$. It has been demonstrated that the columns of $\boldsymbol{A}$ obtained with the convex-NMF are close to the real centroids of clusters. Convex-NMF can be kernelized as well [35]. We implemented the convex-NMF and its kernel version in `convexnmfrule` and `kernelconvexnmf`, respectively.

The basis vectors obtained with the above NMFs are non-orthogonal. Alternatively, *orthogonal NMF* (ortho-NMF) imposes the orthogonality constraint in order to enhance

sparsity [97]. Its formulation is

$$\begin{aligned} &\boldsymbol{X} = \boldsymbol{ASY} + \boldsymbol{E} \\ &\text{s.t. } \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} = \boldsymbol{I}, \quad \boldsymbol{YY}^{\mathrm{T}} = \boldsymbol{I}, \quad \boldsymbol{A}, \boldsymbol{S}, \boldsymbol{Y} \geq 0, \end{aligned} \tag{3.13}$$

where, the input $\boldsymbol{X}$ is non-negative, $\boldsymbol{S}$ absorbs the magnitude due to the normalization of $\boldsymbol{A}$ and $\boldsymbol{Y}$. Function `orthnmfrule` is its implementation in our toolbox. Ortho-NMF is very similar with the *non-negative sparse PCA* (NSPCA) proposed in [98]. The disjoint property on ortho-NMF may be too restrictive for many applications, therefore this property is relaxed in NSPCA. Ortho-NMF does not guarantee the maximum-variance property which is also relaxed in NSPCA. However NSPCA only enforces non-negativity on the basis vectors, even when the training data have negative values. We plan to devise a model in which the disjoint property, the maximum-variance property, the non-negativity and sparsity constraints can be controlled on both basis vectors and coefficient vectors.

There are two efficient ways of applying NMF on data containing missing values. First, the missing values can be estimated prior to running NMF. Alternatively, *weighted-NMF* [99] can be directly applied to decompose the data. Weighted-NMF puts a zero weight on the missing elements and hence only the non-missing data contributes to the final result. An *expectation-maximization* (EM) based missing value estimation during the execution of NMF may not be efficient. The weighted-NMF is given in our toolbox in function `wnmfrule`.

2.3 Results and Discussions

Based on the various implemented NMFs, a number of data mining tasks can be performed via our toolbox. Table 3.2 lists the data mining functionalities we provide in this level. These mining tasks are also described along with appropriate examples.

Clustering and Biclustering

NMF has been applied for clustering. Given data $\boldsymbol{X}$ with multivariate data points in the columns, the idea is that, after applying NMF on $\boldsymbol{X}$, a multivariate data point, say $\boldsymbol{x}_i$ is a non-negative linear combination of the columns of $\boldsymbol{A}$; that is $\boldsymbol{x}_i \approx \boldsymbol{A}\boldsymbol{y}_i = y_{1i}\boldsymbol{a}_1 + \cdots + y_{ki}\boldsymbol{a}_k$. The largest coefficient in the i-th column of $\boldsymbol{Y}$ indicates the cluster this data point belongs to. The reason is that if the data points are mainly composed with the same basis vectors, they should therefore be in the same group. A basis vector is usually viewed as a cluster centroid or prototype. This approach has been used in [75] for clustering

Table 3.2: NMF-based data mining approaches.

Function	Description
NMFCluster	Take the coefficient matrix produced by a NMF algorithm, and output the clustering result.
chooseBestk	Search the best number of clusters based on dispersion coefficients.
biCluster	Biclustering method using one of the NMF algorithms.
featureExtractionTrain	General interface. Using training data, generate the bases of the NMF feature space.
featureExtractionTest	General interface. Map the test/unknown data into the feature space.
featureFilterNMF	On training data, select features by various NMFs.
featSel	Feature selection methods.
nnlsClassifier	The NNLS classifier.
perform	Evaluate the classifier performance.
changeClassLabels01	Change the class labels to be in $\{0, 1, 2, \cdots, C-1\}$ for C-class problem.
gridSearchUniverse	a framework to do line or grid search.
classificationTrain	Train a classifier. Many classifiers are included.
classificationPredict	Predict the class labels of unknown samples via the model learned by classificationTrain.
multiClassifiers	Run multiple classifiers on the same training data.
cvExperiment	Conduct experiment of k-fold cross-validation on a data set.
significantAcc	Check if the given data size can obtain significant accuracy.
learnCurve	Fit the learning curve.
FriedmanTest	Friedman test with post-hoc Nemenyi test to compare multiple classifiers on multiple data sets.
plotNemenyiTest	Plot the crucial-difference diagram of Nemenyi test.
NMFHeatMap	Draw and save the heat maps of NMF clustering.
NMFBicHeatMap	Draw and save the heat maps of NMF biclustering.
plotBarError	Plot bars with STD.
writeGeneList	Write the gene list into a .txt file.
normmean0std1	Normalization to have mean 0 and standard deviation 1.
sparsity	Calculate the sparsity of a matrix.
MAT2DAT	Write a data set from MATLAB into .dat format in order to be readable by other languages.

microarray data and in order to discover tumor subtypes. We have implemented function `NMFCluster` through which various NMF algorithms can be selected. An example is provided in `exampleCluster` file in the folder of our toolbox.

The task of interpreting both the basis matrix and the coefficient is equivalent to simultaneously clustering the rows and columns of matrix $\boldsymbol{X}$. This is biclustering and the interested readers can refer to [100] for an excellent survey on biclustering algorithms and to [87] for a biclustering method based on NMF. We implemented a biclustering approach based on NMF in the `biCluster` function. The biclusters can be visualized via function `NMFBicHeatMap`. We applied NMF to simultaneously grouping the genes and samples of

a leukemia data set [75] which includes tumor samples of three subtypes. The goal is to find strongly correlated genes over a subset of samples. A subset of such genes and a subset of such samples form a bicluster. The heat-map is shown in Figure 3.1. Readers can find the script in `exampleBiCluster` file of our toolbox.

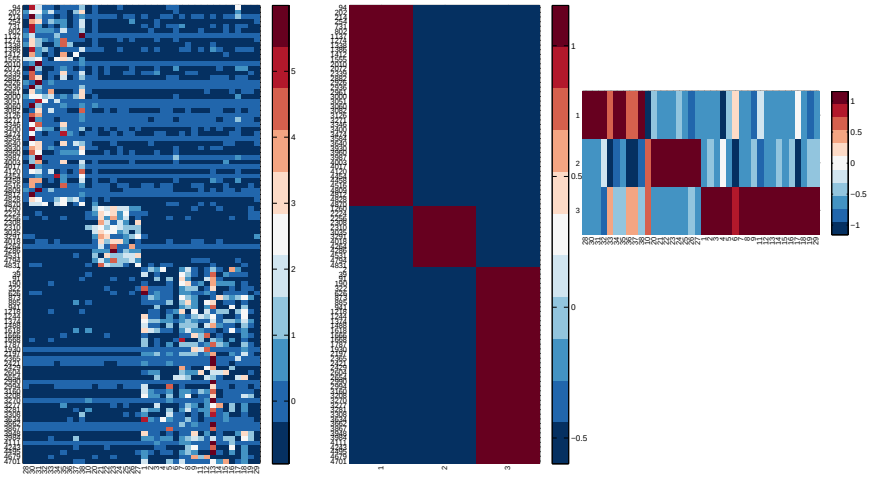


Figure 3.1: Heat map of NMF biclustering result. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix. This is a color figure, thus the readability may be affected if printed in grayscale.

Basis Vector Analysis for Biological Process Discovery

We can obtain interesting and detailed interpretations via an appropriate analysis of the basis vectors. When applying NMF on a microarray data, the basis vectors are interpreted as potential biological processes [45, 38, 1]. In the following, we give one example for finding biological factors on gene-sample data, and two examples on time-series data. Please note they only serve as simple examples. Fine tuning of the parameters of NMF is necessary for accurate results.

First example: We ran our VSMF on the ALLAML gene-sample data of [75] with the settings $k = 3$, $\alpha_1 = 0.01$, $\alpha_2 = 0.01$, $\lambda_1 = 0$, $\lambda_2 = 0.01$, $t_1 = 1$, and $t_2 = 1$. Next, we obtain 81, 37, and 448 genes for the three factors, respectively. As in [38], we then performed *gene set enrichment analysis* (GSEA) by applying Onto-Express [101] on each of these sets of genes. Part of the result is shown in Table 3.3. We can see that the factor-specific genes selected by NMF correspond to some biological processes significantly. Please see file

`exampleBioProcessGS` in the toolbox for details. GSEA can also be done using other tools, such as MIPS [102], GOTermFinder [103], and DAVID [104, 105].

Table 3.3: Gene set enrichment analysis using Onto-Express for the factor specific genes identified by NMF.

Factor 1		Factor 2		Factor 3	
biological process	p-val.	biological process	p-val.	biological process	p-val.
reproduction (5)	0	response to stimulus (15)	0.035	regulation of bio. proc. (226)	0.009
metabolic process (41)	0	biological regulation(14)	0.048	multi-organism proc. (39)	0.005
cellular process (58)	0			biological regulation (237)	0.026
death (5)	0				
developmental process (19)	0				
regulation of biological process (19)	0				

Second example: We used NMF to cluster a time-series data of yeast metabolic cycle in [106]. Figure 3.2 shows the heat-map of NMF clustering, and Figure 3.3 shows the three basis vectors. We used `nmfnnls` function to decompose the data and `NMFHeatMap` to plot the heat-map. The detailed script is given in the `exampleBioProcessTSYeast` file in the toolbox. We can clearly see that the three periodical biological processes correspond exactly to the Ox (oxidative), R/B (reductive, building), and R/C (reductive, charging) processes discovered in [106].

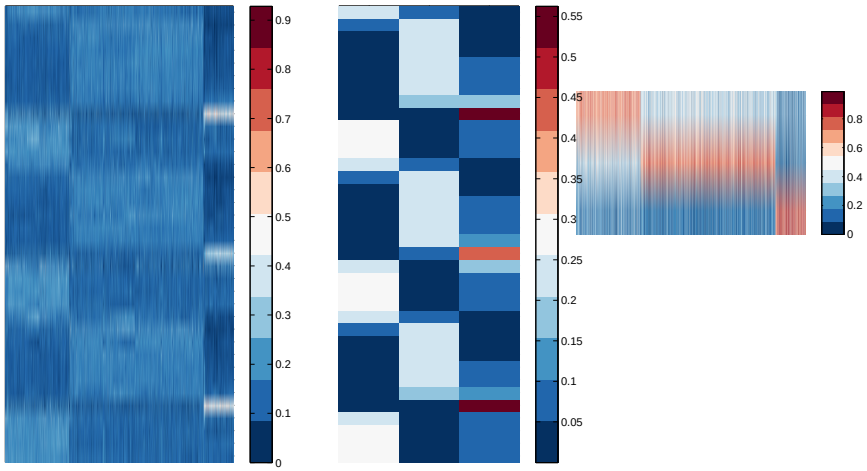


Figure 3.2: Heat map of NMF clustering result on yeast metabolic cycle time-series data. Left: the gene expression data where each column corresponds to a sample. Center: the basis matrix. Right: the coefficient matrix. This is a color figure, thus the readability may be affected if printed in grayscale.

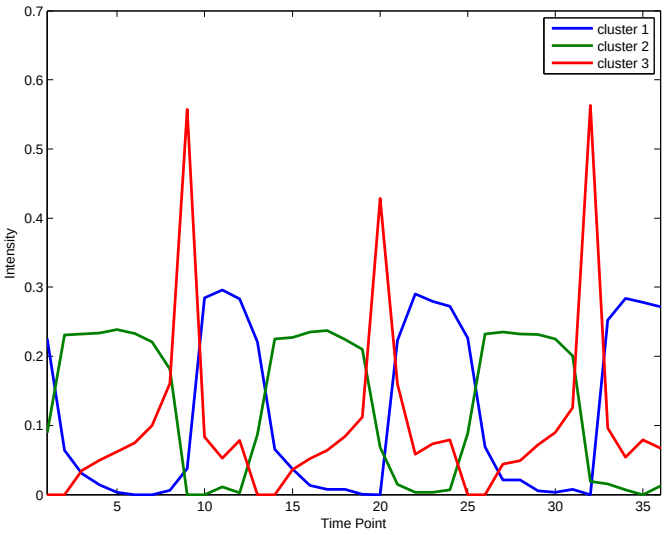


Figure 3.3: Biological processes discovered by NMF on yeast metabolic cycle time-series data. This is a color figure, thus the readability may be affected if printed in grayscale.

Third example: We used NMF to factorize a breast cancer time-series data set, which includes wild type MYCN cell lines and mutant MYCN cell lines [107]. The purpose of this example is to show that NMF is a potential tool to find cancer drivers. One basic methodology is in the following. First, basis vectors are produced applying NMF on a time-series data. Then factor-specific genes are identified by computational or statistical methods. Finally, the regulators of these factor-specific genes are identified from any prior biological knowledge. This data set has 8 time points (0, 2, 4, 8, 12, 24, 36, 48 hr.). Samples at time point zero are untreated. Samples were collected at the subsequent time points after treatment with 4-hydroxytamoxifen (4-OHT). In our computational experiment, we use our VSMF implementation (function `vsmf`). we set $k = 2$. Because this data set has negative values we set $t_1 = 0$ and $t_2 = 1$. We set $\alpha_1 = 0.01$, $\alpha_2 = 0$, $\lambda_1 = 0$, and $\lambda_2 = 0.01$. The basis vectors of both wild-type and mutant data are compared in Figure 3.4. From the wild-type time-series data, we can successfully identify two patterns. The rising pattern corresponds to the induced signature while the falling pattern corresponds to the repressed signature in [107]. It is reported in [107] that the MYC target genes contribute to both patterns. From the mutant time-series, we can obtain two flat processes, which are reasonable. The source code of this example can be found in `exampleBioProcessMYC`. We also recommend the readers to see the methods based on matrix decompositions which are proposed in [1] and [46] and devised for identifying signaling pathways.

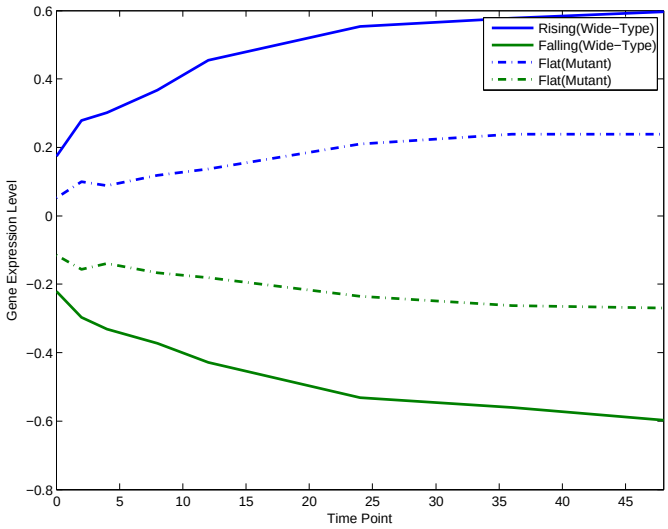


Figure 3.4: Biological processes discovered by NMF on breast cancer time-series data. This is a color figure, thus the readability may be affected if printed in grayscale.

Basis Vector Analysis for Gene Selection

The columns of $\boldsymbol{A}$ for a gene expression data set are called *metasamples* in [75]. They can be interpreted as biological processes, because their values imply the activation or inhibition of some genes. Gene selection aims to find marker genes for disease prediction and to understand the pathways they contribute to. Rather than selecting genes on the original data, the novel idea is to conduct gene selection on the metasamples. The reason is that the discovered biological processes via NMF are biologically meaningful for class discrimination in disease prediction, and the genes expressed differentially across these processes contribute to better classification performance in terms of accuracy. In Figure 3.1 for example, three biological processes are discovered and only the selected genes are shown. We have implemented the information-entropy-based gene selection approach proposed in [38] in function `featureFilterNMF`. We give an example on how to call this function in file `exampleFeatureSelection`. It has been reported that NMF can select meaningful genes, which have been verified with gene ontology analysis. In the future, we are also interested in implementing feature selection methods based on *supervised* NMF.

Feature Extraction

Microarray data and mass spectrometry data have tens of thousands of features but only tens or hundreds of samples. This leads to the issues of *curse of dimensionality*. For

example, it is impossible to estimate the parameters of some statistical models since the number of their parameters grow exponentially as the dimension increases. Another issue is that biological data are usually noisy; which crucially affects the performances of classifiers applied on the data. In cancer study, a common hypothesis is that only a few biological factors (such as the oncogenes) play a crucial role in the development of a given cancer. When we generate data from control and sick patients, the high-dimensional data will contain a large number of irrelevant or redundant information. Orthogonal factors obtained with *principal component analysis* (PCA) may not appropriate in most cases. Since NMF generates non-orthogonal (and non-negative) factors, therefore it is much reasonable to extract important and interesting features from such data using NMF. As mentioned above, training data $\boldsymbol{X}_{m\times n}$, with m features and n samples, can be decomposed into k metasamples $\boldsymbol{A}_{m\times k}$ and $\boldsymbol{Y}_{k\times n}$, that is

$$\boldsymbol{X} \approx \boldsymbol{A}\boldsymbol{Y}_{\text{trace}}, \text{ s.t. } \boldsymbol{A}, \boldsymbol{Y}_{\text{trace}} \geq 0, \tag{3.14}$$

where, $\boldsymbol{Y}_{\text{trace}}$ means that $\boldsymbol{Y}$ is obtained from the training data. The k columns of $\boldsymbol{A}$ span the k-dimensional *feature space* and each column of $\boldsymbol{Y}_{\text{trace}}$ is the representation of the corresponding original training sample in the feature space. In order to project the p unknown samples $\boldsymbol{S}_{m\times p}$ into this feature space, we have to solve the following non-negative least squares problem:

$$\boldsymbol{S} \approx \boldsymbol{A}\boldsymbol{Y}_{\text{uk}}, \text{ s.t. } \boldsymbol{Y}_{\text{uk}} \geq 0, \tag{3.15}$$

where, $\boldsymbol{Y}_{\text{uk}}$ means the $\boldsymbol{Y}$ is obtained from the unknown samples. After obtaining $\boldsymbol{Y}_{\text{trace}}$ and $\boldsymbol{Y}_{\text{uk}}$, the learning and prediction steps can be done quickly in the k-dimensional feature space instead of the m-dimensional original space. A classifier can learn over $\boldsymbol{Y}_{\text{trace}}$, and then predicts the class labels of the representations of unknown samples, that is $\boldsymbol{Y}_{\text{uk}}$.

From the aspect of interpretation, the advantage of NMF over PCA and ICA is that the metasamples are very useful in the understanding of the underlying biological processes, as mentioned above.

We have implemented a pair of functions `featureExtractionTrain` and `featureExtractionTest` including many linear and kernel NMF algorithms. The basis matrix (or, the inner product of basis matrices in the kernel case) is learned from the training data via the function `featureExtractionTrain`, and the unknown samples can be projected onto the feature space via the function `featureExtractionTest`. We give examples of how to use these functions in files `exampleFeatureExtraction` and `exampleFeatureExtractionKernel`.

Figure 3.5 shows the classification performance of SVM *without* dimension reduction

and SVM *with* dimension reduction using linear NMF, kernel NMF with *radial basis function* (RBF) kernel, and PCA on two data sets, SRBCT [80] and Breast [31]. Since ICA is computationally costly, we did not include it in the comparisons. The bars represent the averaged four-fold cross-validation accuracies using *support vector machine* (SVM) as classifier over 20 runs. We can see that NMF is comparable to PCA on SRBCT, and is slightly better than PCA on Breast data. Also, with only few factors, the performance after dimension reduction using NMF is at least comparable to that without using any dimension reduction. As future work, supervised NMF will be investigated and implemented in order to extract discriminative features.

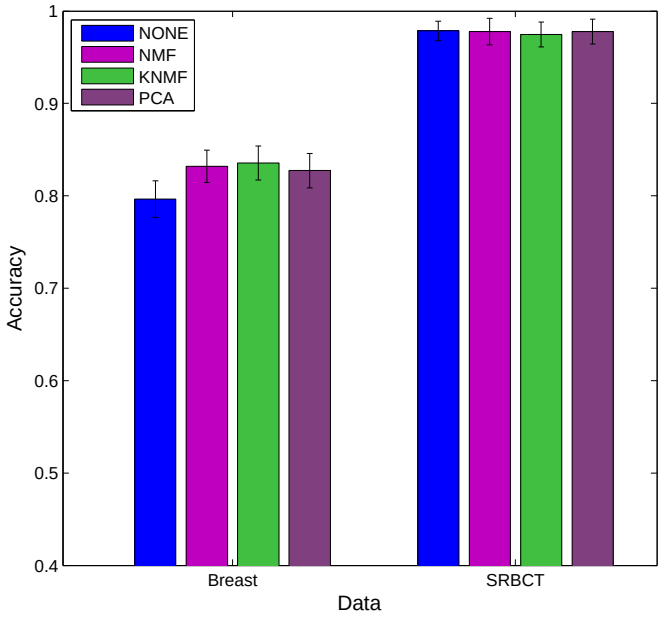


Figure 3.5: Mean accuracy and standard deviation results of NMF-based feature extraction on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

Classification

If we make the assumption that every unknown sample is a sparse non-negative linear combination of the training samples, then we can directly derive a classifier from NMF. Indeed, this is a specific case of NMF in which the training samples are the basis vectors. Since the optimization process within NMF is a NNLS problem, we call this classification

approach the *NNLS classifier* [18]. A NNLS problem is essentially a quadratic programming problem as formulated in Equation (3.9), therefore, only the inner products are needed for the optimization. We thus can naturally extend the NNLS classifier to kernel version. Two features of this approach are that: i) the sparsity regularization help avoid overfitting the model; and ii) the kernelization allows a dimension-free optimization and also linearizes the non-linear complex patterns within the data. The implementation of the NNLS classifier is in file `nnlsClassifier`. Our toolbox also provides many other classification approaches including SVM classifier. Please see file `exampleClassification` for demonstration. In our experiment of four-fold cross-validation, accuracies of 0.7865 and 0.7804 are respectively obtained with linear and kernel (RBF) NNLS classifier on Breast data set. They achieved accuracies of 0.9762 and 0.9785, respectively, over SRBCT data.

Biological data are usually noisy and sometimes contain missing values. A strength of the NNLS classifier is that it is robust to noise and to missing values, making NNLS classifiers quite suitable for classifying biological data [18].

In order to show its robustness to noise, we added a Gaussian noise on SRBCT. The Gaussian distribution has mean 0 and variance from 0 to 4 with increment 0.5. Figure 3.6 illustrates the results of NNLS, SVM, and *1-nearest neighbor* (1-NN) classifiers using this noisy data. It can be seen that as the noise increases, NNLS outperforms SVM and 1-NN significantly.

To deal with the missing value problem, three strategies are usually used: incomplete sample or feature removal, missing value imputation (i.e., estimation), and ignoring missing values. Removal methods may delete important or useful information for classification and particularly when there is a large percentage of missing values in the data. Imputation methods may create false data depending on the magnitude of the true estimation errors. The third method ignores missing values using the weighting strategy during classification. Our approach in dealing with the missing value problem is also to ignore them. The NNLS optimization needs only the inner products of pairs of samples. Thus, when computing the inner product of two samples, say $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we normalize them to have unit l_2-norm using only the features present in both samples, and then we take their inner product. As an example, we randomly removed between 10% to 70% of data values in STBCT data. Using such incomplete data, we compared our method with the zero-imputation method (that is, estimating all missing values as 0). In Figure 3.7, we can see that the NNLS classifier using our missing value approach outperforms the zero-imputation method in the case of large missing rate. Also, the more sophisticated k-nearest neighbor imputation (KNNimpute) method [108] will fail on data with in high percentage of missing values.

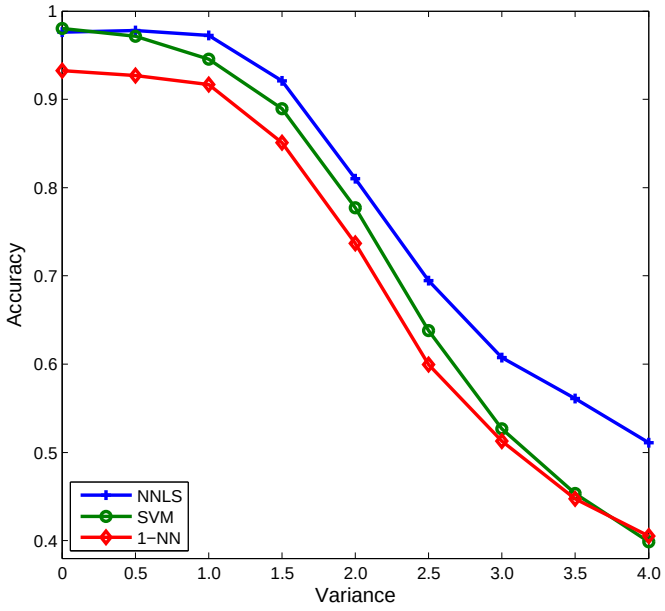


Figure 3.6: The mean accuracy results of NNLS classifier for different amount of noise on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale.

Statistical Comparison

The toolbox provides two methods for statistical comparisons and evaluations of different methods. The first is a two-stage method proposed in [67]. The importance of this method is that it can estimate the data-size requirement for attaining a significant accuracy and extrapolate the performance based on the current available data. Generating biological data is usually very expensive and thus this method can help researchers to evaluate the necessity of producing more data. At the first stage, the minimum data size required for obtaining a significant accuracy is estimated. This is implemented in function `significantAcc`. The second stage is to fit the learning curve using the error rates of large data sizes. It is implemented in function `learnCurve`. In our experiments, we have found that the NNLS classifier usually requires fewer number of samples for obtaining a significant accuracy. For example on SRBCT data, NNLS requires only 4 training samples while SVM needs 19 training samples. The fitted learning curves of NNLS and SVM classifiers are shown in Figure 3.8. We provide an example of how to plot this figure in file `exampleFitLearnCurve`.

The second method is the nonparametric Friedman test coupled with the post-hoc Nemenyi test to compare multiple classifiers over multiple data sets [85]. It is difficult to draw

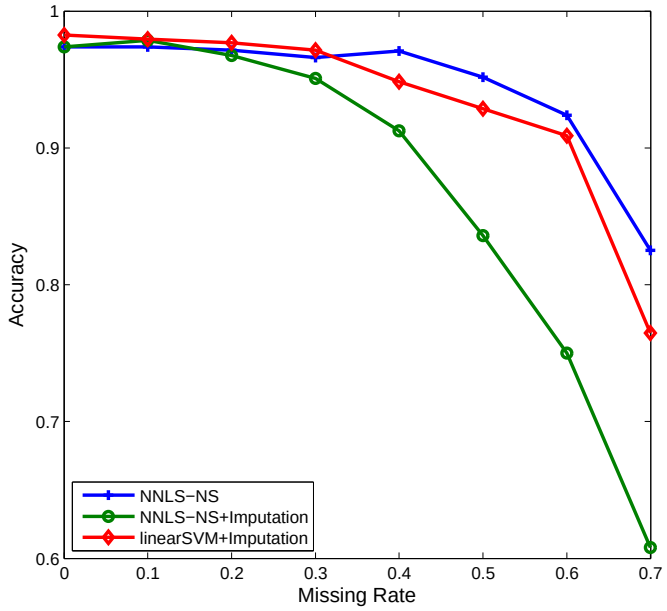


Figure 3.7: The mean accuracy results of NNLS classifier for different missing value rates on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale.

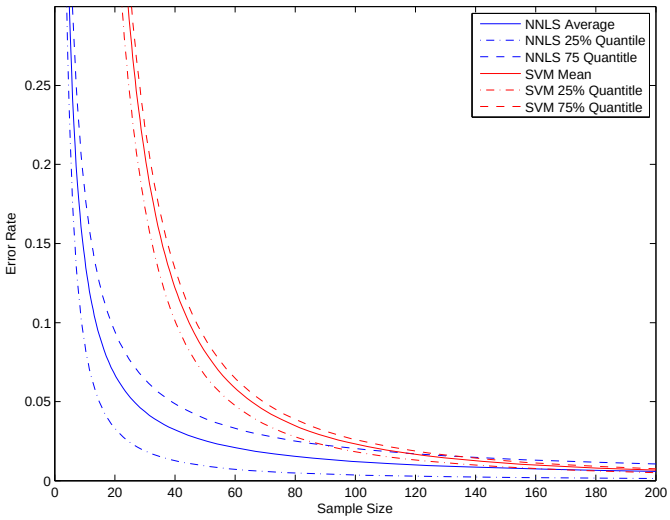


Figure 3.8: The fitted learning curves of NNLS and SVM classifiers on SRBCT data. This is a color figure, thus the readability may be affected if printed in grayscale.

an overall conclusion if we compare multiple approaches in a pairwise fashion. The Friedman test has been recommended in [85] because it is simple, safe and robust, compared

with parametric tests. It is implemented in function `FriedmanTest`. The result can be presented graphically using the *crucial difference* (CD) diagram as implemented in function `plotNemenyiTest`. CD is determined by significance level α. Figure 3.9 is an example of the result of the Nemenyi test for comparing 8 classifiers over 13 high dimensional biological data sets. This example can be found in file `exampleFriedmanTest`. If the distance of two methods is greater than the CD then we conclude that they differ significantly.

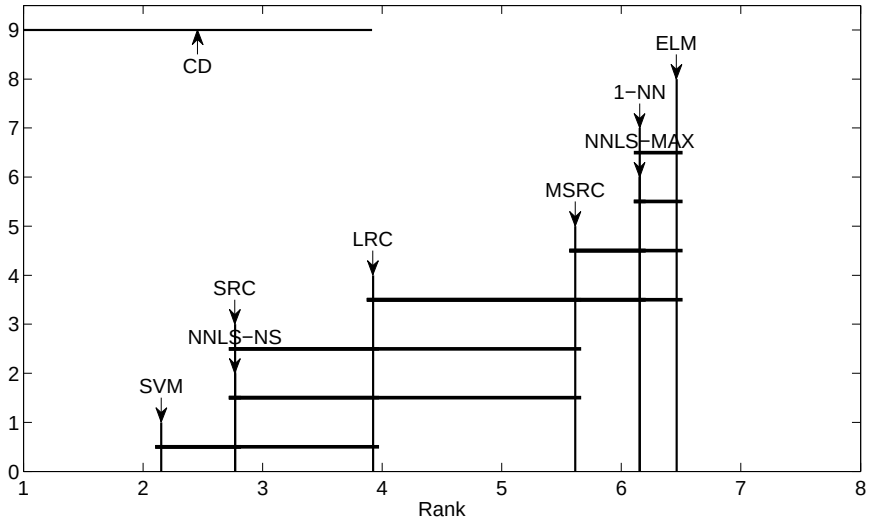


Figure 3.9: Nemenyi test comparing 8 classifiers over 13 high dimensional biological data ($\boldsymbol{\alpha = 0.05}$).

3 The Sparse Representation Toolbox

3.1 Introduction

We have implemented all the sparse representation methods mentioned in the previous section. We pack these implementations all together in our *sparse representation toolbox* (SR toolbox). We separate our code into basic level and advanced level. The basic level is mainly composed of sparse coding models, dictionary learning models, and the corresponding optimization. The advanced level includes kernel classification and feature extraction methods based on sparse representation. We introduce both levels in the subsequent sections. Examples of using these functions can be found in the folder of the source code. The source code of this toolbox can be downloaded at `https://sites.google.com/site/sparseReptool` and `http://cs.uwindsor.ca/~li11112c/sr`.

3.2 Implementations

The Basis Implementation

We summarize our implementations of the basic sparse representation techniques in Table 3.4. Functions `KSRSC` and `KSRDL` are the generic interfaces of kernel sparse coding and dictionary learning. In files `exampleKSRSC`, and `exampleKSRDL`, we provide examples of how to uses these functions, respectively. Function `vsmf` is more general, as has been introduced in the NMF toolbox. In Table 3.4, we also list our implementations of active-set, interior-point, proximal, and decomposition methods for sparse coding. In file `exampleOptSC`, examples are given to show usages of these optimization functions.

Table 3.4: Methods of sparse representation.

Function	Description
KSRSC	Kernel sparse coding methods including l_1LS, NNLS, and l_1NNLS.
KSRDL	The generic kernel dictionary learning framework.
vsmf	Versatile sparse matrix factorization optimized by NNQP and l_1QP.
l1QPActiveSet	The active-set method for single or multiple l_1QP problem.
NNQPActiveSet	The active-set method for single or multiple NNQP problem.
l1QPIP	The interior-point method for single l_1QP problem.
l1QPIPMulti	The interior-point method for multiple l_1QP problem.
NNQPIP	The interior-point method for single NNQP problem.
NNQPIPMulti	The interior-point method for multiple NNQP problem.
l1QPProximal	The proximal method for single l_1QP problem.
l1QPProximalMulti	The proximal method for multiple l_1QP problem.
l1QPSMO	The SMO method for single l_1QP problem.
l1QPSMOMulti	The SMO method for multiple l_1QP problem.
NNQPSMO	The SMO method for single NNQP problem.
NNQPSMOMulti	The SMO method for multiple NNQP problem.
computeKernelMatrix	Compute the kernel matrix $k(\boldsymbol{A}, \boldsymbol{B})$ given a kernel function.
sparsity	Computer the sparsity of a matrix.
normalizeKernelMatrix	Normalize kernel matrices to let each sample have unit l_2-norm.

The Advanced Implementation

The advanced level of our implementation consists of machine learning applications based on sparse representation. The breakdown of these applications is given in Table 3.5. First of all, using function `KSRSCClassifier`, one can conduct kernel-sparse-coding based classification. The l_1LS, NNLS, l_1LS models with different optimization algorithms and kernels can be specified by the input of `KSRSCClassifier`. Function `computeMetaSample` implements the training of MNNLS, that is learning the sub-dictionaries. The prediction

of MNNLS can be conducted by `KSRSCClassifier`. Function `lrc` is the implementation of kernel *linear regression classification* (LRC) approach which extends the original LRC method [109]. Functions `nearestCentroidTrain` and `nearestCentroidPredict` implement the training and prediction phases of the kernel *nearest centroid* (NC) method. Functions `classificationTrain` and `classificationPredict` are the unified interfaces of these classifiers. Users can easily add their own classifiers in these interfaces. Function `multiClassifiers` allows users to choose multiple classifiers. Cross-validation can be conducted by function `cvExperiment`. The examples of all these methods are given in file `exampleClassification`.

Second, function `featureExtractionTrain` is the unified interface of the training of feature extraction based on dictionary learning. Function `featureExtractionTest` is used after `featureExtractionTrain` to project the unknown samples to the feature space learned by `featureExtractionTrain`. We provide an example of feature extraction in file `exampleFeatExtr`.

Table 3.5: Sparse-representation based machine learning methods.

Function	Description
KSRSCClassifier	The classification approach based on kernel sparse coding.
computeMetaSample	The training of sub-dictionary learning.
lrc	Kernel linear regression classification.
nearestCentroidTrain	Train the kernel nearest centroid classifier.
nearestCentroidPredict	Predict the class labels of unknown samples by the model trained by nearestCentroidTrain.
classificationTrain	Train a classifier. Many classifiers are included.
classificationPredict	Predict the class labels of unknown samples via the model learned by classificationTrain.
multiClassifiers	Run multiple classifiers on the same training data.
cvExperiment	Conduct experiment of k-fold cross-validation on a data set.
featureExtractionTrain	General interface. Using training data, generate the bases of the SR feature space.
featureExtractionTest	General interface. Map the test/unknown data into the feature space.
subspace	The nearest subspace rule used in KSRSCClassifier.
knnrule	The weighted k-nearest neighbor rule used in KSRSCClassifier.
leaveMOut	Leave m out.
changeClassLabels01	Change the class labels to be in $\{0, 1, 2, \cdots, C-1\}$ for C-class problem.
perform	Compute the performance of classification.

3.3 Conclusions

We have developed the NMF toolbox and SR toolbox to accommodate our implementations of sparse representation techniques. The basic level allows the machine learning researchers to devise new methods based on the basic techniques. Researchers from the other areas, such as signal processing and computer vision, can also conveniently apply them. The advanced level facilitates the bioinformaticians to conduct analysis including clustering, feature extraction, feature selection, and classification. Examples are provided to demonstrate the usages of these functionalities.

We mention some of our future works below. First, we will include more NMF algorithms such as nsNMF, LS-NMF, and supervised NMF. Second, we are very interested in implementing and speeding up the Bayesian decomposition method which is actually a probabilistic NMF introduced independently in the same period as the standard NMF. Third, we would like to implement Markov chain Monte Carlo method for the optimization of NMF and sparse representation. Finally, we plan to realize a supervised dictionary learning based on Bayesian regression.

Publications

1. The NMF toolbox was published in [*Y. Li and A. Ngom, "The non-negative matrix factorization toolbox for biological data mining," BMC Source Code for Biology and Medicine, vol. 8, pp. 10, 2013.*].

2. The SR toolbox has not been formally published yet. It is available online [`https://sites.google.com/site/sparsereptool`].

Chapter 4

Tensor Decompositions and Kernel Methods for The Classification of Gene-Sample-Time Data [1]

1 Introduction

With the recent advances in microarray technology, the expression levels of genes with respect to samples can be monitored synchronically over a series of time points. Such microarray data have three types of variables, genes, samples, and time points. Thus, they are tensor data of order three and are termed *gene-sample-time* (GST) microarray data or GST data for short. In the literature, GST data are known as three-dimensional (3D) data. In order to avoid any confusion with the dimensionality of a vector in linear algebra and multivariate statistics, we follow the definition in tensor (or multilinear) algebra and refer to GST data as order three tensor data, or three-way data. We should clarify that dimensionality refers to the number of features of a sample rather than the number of orders of a single value in a data set, therefore the dimensionality of a sample in a GST data set is the number of genes times the number of time points. Figure 4.1 gives an example of a GST data set in tensor and heatmap representations.

Machine learning and data mining approaches are among the main tools to analyzing GST data. For example, by applying classification and gene selection methods, candidate marker genes are selected in [112] from the IFNβ GST data. They can help to discover the pathway of multiple sclerosis. By applying biclustering and triclustering methods, one can

[1] This chapter is based on our publications [110], [18], and [111].

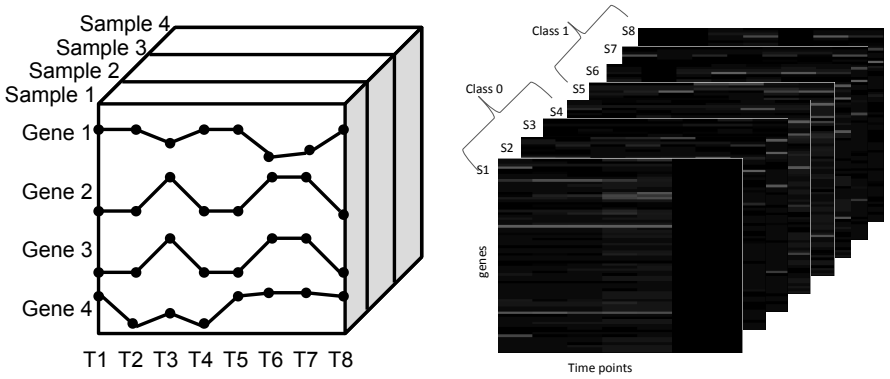


Figure 4.1: A three-way tensor representation (left) and a heatmap representation (right) of a GST data set. The black columns in the heatmap representation correspond to missing time points.

identify subpatterns of genes and samples. It has been well-known that subtypes of breast cancer can be identified by biclustering [113]. Classification techniques can be applied on GST data for the diagnosis of diseases, and the prognosis of an on-going treatment. If a computational method suggests that the patient wouldn't respond well to the current treatment based on the time-series data, then the therapy must be revised or stopped, as many treatments have severe side effects.

Current approaches of analyzing two-way microarray data, such as time-series microarrays (gene-time data) or tissue microarrays (gene-sample data) include supervised or unsupervised learning and data mining methods. However, two-way approaches, such as clustering models for instance, may not be suitable to describe the relationships between the three variables in GST data. They cannot be directly generalized or extended for the three-way GST data. Current GST data contain many missing values in all the three directions. They also contain a large number of genes with a very few number of samples and time-points, and thus they require efficient and effective methods for tackling the potential *curse of dimensionality* rising during learning a computational model. In general, if the number of parameters is much larger than the number of training samples, and grows rapidly even exponentially as the number of dimensions increases, the curse of dimensionality occurs.

Contributions: In this chapter, we propose sparse tensor decomposition methods and kernel methods for the dimensionality reduction and classification of gene-sample-time data. Our contributions are the following:

1. We propose to apply the sparse tensor decomposition method, higher-order non-

negative matrix factorization, to reduce the dimensionality of the gene-sample-time data efficiently.

2. We apply kernel sparse coding and kernel dictionary learning methods for classifying matrix samples and extract vectorial features, respectively.

The rest of this chapter is organized as follows. We first propose our tensor decomposition based dimension reduction method for classifying GST data in Section 2. After that, we propose to use kernel sparse coding, and kernel dictionary learning for the classification of GST data in Section 3. In Appendix A, we shortly introduce the tensor algebra.

2 Tensor Decomposition Methods for Classification

Classification techniques for GST data can be used for diagnosis and prognosis. In this section, we shall introduce non-kernel classification and non-kernel feature extraction approaches for GST data including hidden Markov models and tensor factorizations. In the next section, we introduce kernel approaches. Unlike clustering, in the context of classification, a GST data set is usually represented by a *Gene* $\times$ *Time* $\times$ *Sample* tensor. In the literature, the IFNβ data [112] is one of the most popular data sets to test the performance of methods for GST data analysis. *Interferon beta* (IFNβ) is a protein used for treating patients afflicted with multiple sclerosis (MS), among other diseases. Some MS patients who received IFNβ therapy do not respond well to the drug and the reasons are still not clear [114]. Medical researchers are seeking for genomic reasons via high-throughput data analysis. Baranzini *et al.* [112], among others researchers, applied Bayesian learning methods on a clinical microarray data set to determine pairs or triplets of genes that can discriminate between bad and good IFNβ responders. This data set is available online as the supplementary material of [112]. The initial data set is a GST data sampled from 53 MS patients who were initially treated with equal dose of IFNβ over a time period. This initial data set contains the expression measurements for 76 genes at 7 time points (0, 3, 6, 9, 12, 18 and 24 months) for each patient, with 31 patients responding well and the remaining 22 responding badly to the treatment. This data set contains genes with missing expression measurements at some time points. Those genes and corresponding samples were removed from our analysis, and hence, the resulting "complete" data contains 53 genes and 27 samples (18 good responders and 9 bad responders). In this chapter, we denote this data set as IFNβ, and use it as the working data set for all the classification methods discussed below.

2.1 Related Works

The authors of [112] proposed an *integrated Bayesian inference system* (IBIS) to select triplets of genes for classifying IFNβ samples but using only the first time point, and thus did not benefit from (nor consider) the full GST data. In [115], *generative hidden Markov models* (GenHMMs) and *discriminative HMMs* (DiscHMMs) approaches were devised for classifying IFNβ samples. Samples from the same class are used to train a GenHMM, whereas samples from all classes are used to train a DiscHMM; then a test sample is assigned to a class based on the maximum conditional likelihood. The Baum-Welch algorithm is used to estimate the parameters of the models. For DiscHMMs, backward gene selection is first performed to find a small number of discriminative genes before training the models.

The authors of [116] proposed a robust constrained mixture estimation approach to classify the IFNβ data. This approach combines the constrained clustering method with a mixture estimation classification framework. Subdivision of classes and mislabeled samples can be investigated by this approach. During training, negative constraints were restricted on pairs of samples. The constrained mixture model, with linear HMMs, as components, is optimized by an EM algorithm. The supervised version of this approach (*HMMConst*) only uses the training set in the estimation of parameters, while the semi-supervised version (*HMMConstAll*) uses all the data. The emission probability for each state is modeled by a mixture of multivariate Gaussians for patient expression values, noise, and missing values, respectively. In order to select genes contributing to classification, a HMM-based gene ranking method is used. Each component of the mixture model is assigned to a class. When testing, a test sample is assigned to a class according to the maximum entry in their posterior distribution.

The classification performance of the GenHMMs and DiscHMMs are compared on IFNβ data, while the implementations of HMMConst and HMMConstAll [116] are not available, we thus can not investigate their performance. Nine-fold cross-validation was employed to split the whole data into training sets and test sets. It was rerun for 20 times, and the means and standard deviations of specificity, sensitivity, and accuracy are given in Table 4.1. The accuracy is defined as the ratio of the number of correctly predicted test samples to the total number of test samples. Good responders are treated as "negative", while bad responders are "positive". Therefore, specificity is the prediction accuracy of the good responders, that is, the ratio of the number of correctly predicted good responders to the total number of good responders, while sensitivity is that of the bad responders. The parameter for GenHMMs and DiscHMMs is the number of selected genes; absence of such parameter means gene selection was not used. First of all, we can see that both methods

Table 4.1: Classification performance of hidden Markov models on complete IFNβ data.

Method	Parameter	Specificity	Sensitivity	Accuracy
GenHMMs	-	0.8611±0.036	0.5556±0.000	0.7593±0.044
DiscHMMs	-	0.8611±0.036	0.5556±0.000	0.7593±0.044
GenHMMs	7	0.8611±0.063	0.5611±0.008	0.7611±0.047
DiscHMMs	7	0.8611±0.063	0.5611±0.008	0.7611±0.047

have low sensitivity and high specificity. This is consistent with the clinical result that the response to IFNβ is difficult to predict. Second, the average prediction accuracy of both methods are promising. Third, by selecting genes, the accuracy can be slightly improved.

2.2 Tensor Decomposition for Feature Extraction

We propose to use tensor decomposition to reduce the dimensionality of GST data before classification. A few new features can be extracted by *Linear dimension reduction* (LDR) [117] methods, in order to capture useful information for classification or clustering. Each of the new features is a linear combination of the original features. A transformation matrix projects the original samples into a new space, termed *feature space*. A sample in the feature space is a *representation* of the corresponding original sample. Taking NMF for example, a non-negative training set $\boldsymbol{X}^{\text{train}}$ with m genes and n samples of a gene-sample data can be decomposed into a non-negative basis matrix $\boldsymbol{A}^{\text{train}}$ and a non-negative coefficient matrix $\boldsymbol{Y}^{\text{train}}$, that is

$$\boldsymbol{X}^{\text{train}}_{m\times n} \approx \boldsymbol{A}^{\text{train}}_{m\times r}\boldsymbol{Y}^{\text{train}}_{r\times n}, \quad \boldsymbol{X}^{\text{train}}, \boldsymbol{A}^{\text{train}}, \boldsymbol{Y}^{\text{train}} \geq 0, \tag{4.1}$$

where $r < \min\{m, n\}$ is the number of dimensions of the feature space. Each column of $\boldsymbol{Y}^{\text{train}}$ is a representation of the corresponding original sample in the feature space spanned by the columns of $\boldsymbol{A}^{\text{train}}$. In the feature space, a new feature is a linear combination of the original n genes. A sample in the original space can be mapped into the feature space by the transformation matrix $(\boldsymbol{A}^{\text{train}})^T$. The dimension of the feature space is much lower than that of the original space. LDR methods extend into *multilinear dimension reduction* (MLDR) methods in tensor algebra. Before reading the following content, the reader is referred to Appendix A for an introduction to tensor notations, tensor operations, and tensor factorizations including PARAFAC decomposition and Tucker decomposition. Tensor decomposition has been applied in the analysis of high-order microarray data in [118] and [119]. We focus on the MLDR method for GST data below. Let $\boldsymbol{\mathcal{X}}$ be a training set, from a GST data set, with I_1 genes, I_2 time points, and I_3 samples. Factorizing the

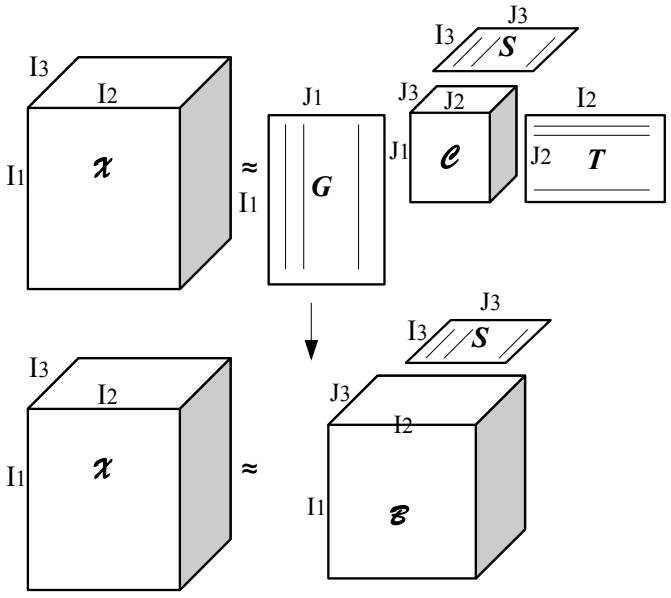


Figure 4.2: From Tucker3 decomposition (top) to Tucker1 decomposition (bottom).

GST data $\mathcal{X}$ by Tucker3 decomposition as in Equation (A.6), we can obtain

$$\mathcal{X} \approx \mathcal{B} \times_3 \boldsymbol{S} = [\![\mathcal{B}; \boldsymbol{I}_G, \boldsymbol{I}_T, \boldsymbol{S}]\!], \tag{4.2}$$

where $\mathcal{B} = \mathcal{C} \times_1 \boldsymbol{G} \times_2 \boldsymbol{T}$, $\boldsymbol{I}_G$ and $\boldsymbol{I}_T$ are identity matrices of sizes $I_1 \times I_1$ and $I_2 \times I_2$, respectively, $\boldsymbol{S}$ is of size $I_3 \times J_3$. The Tucker3 decomposition and Tucker1 decomposition are illustrated in Figure 4.2.

Making use of multilinear operations, we have

$$\begin{aligned} \boldsymbol{X}_{(1)} &\approx \boldsymbol{I}_G \boldsymbol{B}_{(1)} (S \otimes \boldsymbol{I}_T)^T \\ &= \boldsymbol{I}_G [\boldsymbol{B}_1, \boldsymbol{B}_2, \cdots, \boldsymbol{B}_{J_3}][\boldsymbol{s}_1 \otimes \boldsymbol{I}_T, \boldsymbol{s}_2 \otimes \boldsymbol{I}_T, \cdots, \boldsymbol{s}_{J_3} \otimes \boldsymbol{I}_T]^T \\ &= \sum_{r=1}^{J_3} \boldsymbol{I}_G \boldsymbol{B}_r (\boldsymbol{s}_r \otimes \boldsymbol{I}_T)^T, \end{aligned} \tag{4.3}$$

where $\boldsymbol{B}_r = \mathcal{B}(:,:,r)$ is the r-th frontal slice of $\mathcal{B}(:,:,r)$, and $\boldsymbol{s}_r$ is the r-th column vector of $\boldsymbol{S}$. Via tensorization, we have

$$\mathcal{X} \approx \sum_{r=1}^{J_3} \boldsymbol{B}_r \times_3 \boldsymbol{s}_r \,, \tag{4.4}$$

which approximates the GST data, $\boldsymbol{\mathcal{X}}$, by the summation of J_3 tensors. We can see it more clearly by the matrix formulation as follows:

$$\begin{aligned}\boldsymbol{X}_{(1)} &\approx \boldsymbol{I}_G \boldsymbol{B}_{(1)} (\boldsymbol{S} \otimes \boldsymbol{I}_T)^T \\ &= [\boldsymbol{B}_1, \boldsymbol{B}_2, \cdots, \boldsymbol{B}_{J_3}] \begin{bmatrix} \boldsymbol{s}_{11}\boldsymbol{I}_T & \cdots & \boldsymbol{s}_{I_3 1}\boldsymbol{I}_T \\ \vdots & \vdots & \vdots \\ \boldsymbol{s}_{1J_3}\boldsymbol{I}_T & \cdots & \boldsymbol{s}_{I_3 J_3}\boldsymbol{I}_T \end{bmatrix}. \end{aligned} \tag{4.5}$$

Thus, the k-th frontal slice of $\boldsymbol{\mathcal{X}}$, that is, the k-th sample, can be fitted by the summation of the frontal slices of $\boldsymbol{\mathcal{B}}$:

$$\boldsymbol{X}_{(1)k} \approx \sum_{r=1}^{J_3} \boldsymbol{B}_r \boldsymbol{s}_{kr}, \tag{4.6}$$

where the coefficients are in the k-th row (denoted by $\boldsymbol{s}_k$) of $\boldsymbol{S}$.

We can see that $\boldsymbol{\mathcal{B}}$ is the basis tensor for the samples and $\boldsymbol{S}$ is the encoding matrix. We can define the matrix space spanned by $\boldsymbol{\mathcal{B}}$ as *feature space*, and $\boldsymbol{s}_k$ as the representation of the k-th sample in the feature space. In the sense of feature extraction, these matrix slices of $\boldsymbol{\mathcal{B}}$ are the *features*. This reduces the original sample slice to a vector $\boldsymbol{s}_k$ in the feature space. Figure 4.3 illustrates the idea of tensor-factorization-based feature extraction. Additionally, it is noted that $\boldsymbol{\mathcal{C}} \times_2 \boldsymbol{T} \times_3 \boldsymbol{S}$ and $\boldsymbol{\mathcal{C}} \times_1 \boldsymbol{G} \times_3 \boldsymbol{S}$ are the basis tensors for genes and time points, respectively. If the training set is decomposed by *high-order non-negative matrix factorization* (HONMF), the extracted non-negative features would be interpretable, and a sample will be an additive summation of the features.

In the test phase, each test sample $\boldsymbol{Y}_l$ is projected onto the feature space. $\boldsymbol{Y}_l$ is a linear combination of the basis matrices in $\boldsymbol{\mathcal{B}}$:

$$\boldsymbol{Y}_l = \sum_{r=1}^{J_3} \boldsymbol{B}_r \alpha_r, \tag{4.7}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_{J_3}]^T$ is the representation of $\boldsymbol{Y}_l$ in the feature space. Finding $\boldsymbol{\alpha}$ is equivalent to solving the following generalized least squares problem:

$$\min_{\boldsymbol{\alpha}} \|\boldsymbol{Y}_l - \sum_{r=1}^{J_3} \boldsymbol{B}_r \alpha_r\|_F^2. \tag{4.8}$$

The general solution to this problem is $\alpha_r = \frac{<\boldsymbol{Y}_l, \boldsymbol{B}_r>}{<\boldsymbol{B}_r, \boldsymbol{B}_r>}$ [120], where $< \bullet, \bullet >$ is the inner product of two matrices. For different test samples, we put the $\boldsymbol{\alpha}$'s in the corresponding rows of a

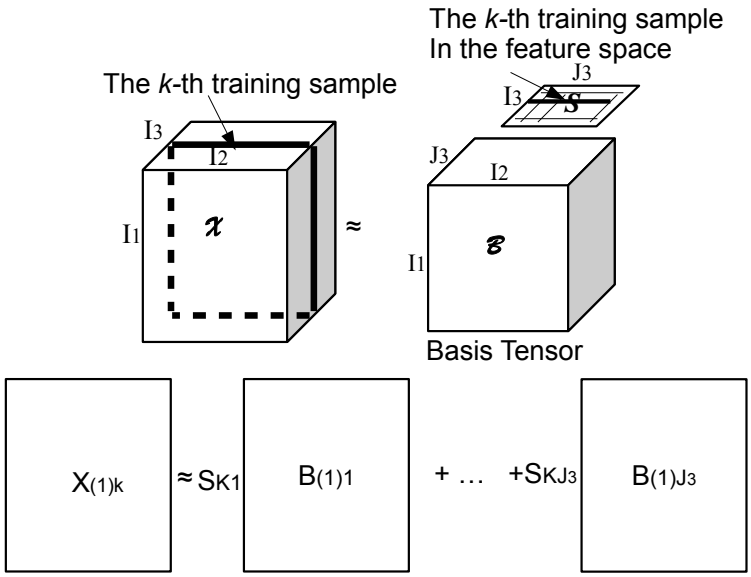


Figure 4.3: Tensor factorization based feature extraction.

coefficient matrix $\boldsymbol{A}$. We employed three Tucker models including *high-order singular value decomposition* (HOSVD), *high-order orthogonal iterations* (HOOI), and HONMF (see Appendix A for their introductions). The *unsupervised* MLDR methods above based on these three Tucker models are denoted by uHOSVDls, uHOOIls, and uHONMFls, respectively.

Alternatively, given the test samples $\mathcal{Y}$, we can fix $\mathcal{C}$, $\boldsymbol{G}$, and $\boldsymbol{T}$ to calculate the coefficient matrix $\boldsymbol{A}$ of $\mathcal{Y}$. We need to find $\boldsymbol{A}$ that satisfies

$$\mathcal{Y} \approx \mathcal{B} \times_3 \boldsymbol{A} = \mathcal{C} \times_1 \boldsymbol{G} \times_2 \boldsymbol{T} \times_3 \boldsymbol{A}. \tag{4.9}$$

For HOSVD and HOOI, the mode matrices, $\boldsymbol{G}$, $\boldsymbol{T}$, and $\boldsymbol{A}$ are column-wise orthonormal. As seen from Appendix A, we can obtain $\boldsymbol{A}$ by using SVD on $\boldsymbol{Z}_{(3)}$. $\boldsymbol{Z}_{(3)}$ is matricized from $\mathcal{Z}$ in mode 3, which is calculated by the following equation:

$$\mathcal{Z} = \mathcal{Y} \times_1 \boldsymbol{G}^T \times_2 \boldsymbol{T}^T. \tag{4.10}$$

For HONMF, the constraint on the mode matrices is non-negativity rather than orthogonality. Instead of solving the non-negativity constrained equation similar to Equation (4.10), $\boldsymbol{A}$ can be rapidly obtained using the update rules of the HONMF algorithm. We can iteratively update $\boldsymbol{A}$ only, while keeping $\mathcal{C}$, $\boldsymbol{G}$, and $\boldsymbol{T}$ constant. If this method is used for HONMF and Equation (4.10) is used for HOSVD and HOOI, then the resulting algorithms

are denoted by uHONMFtf, uHOSVDtf, and uHOOItf, respectively.

Once $\boldsymbol{A}$ is obtained, we do not need to learn on the training samples and classify the test samples represented by the matrices. Instead, any classifier can be trained on the rows of $\boldsymbol{S}$ and classify the rows of $\boldsymbol{A}$. That is, the classification is conducted in the feature space.

Although the decomposition methods described above are unsupervised dimensionality reduction techniques, they can be modified to perform in a supervised manner, i.e. such that class information is taken into account during decomposition. Let m be the number of distinct class labels in the data. The idea is to first partition the training set into m subsets $\boldsymbol{\mathcal{X}}^1$, $\boldsymbol{\mathcal{X}}^2$, $\cdots$, $\boldsymbol{\mathcal{X}}^m$, where each subset $\boldsymbol{\mathcal{X}}^i$ contains only samples of class i. Next, m core tensors $\boldsymbol{\mathcal{B}}^1$, $\boldsymbol{\mathcal{B}}^2$, $\cdots$, $\boldsymbol{\mathcal{B}}^m$ are obtained through decomposition using Equation (4.2). The resulting basis matrices are then normalized using the Frobenius norm. For a normalized test sample, we fit it using these basis tensors, respectively, through Equation (4.8). This sample is assigned to the class which obtains the minimal fitting residual. For simplicity, we denote the supervised version of HOSVD, HOOI, and HONMF based classification methods by sHOSVD, sHOOI, sHONMF. This supervised decomposition approach is described in [121] for handwritten recognition using HOSVD.

We implemented the above tensor-based approaches using MATLAB. Our implementation is based on *The N-way Toolbox for MATLAB* [122] and *Algorithms for SN-TUCKER* [123]. SN-TUCKER is in fact HONMF. We used the well-known k-nearest neighbor classifier [2] with the Euclidean distance in the classification after our unsupervised feature extraction methods. The parameter of the tensor decomposition based approaches are rank-(J_1, J_2, J_3), and grid search is performed to find the values of J_1, J_2, J_3 which give best classification performance. The parameters of all models were selected by grid search. We used nine-fold cross-validation, because there are only 27 samples and in each fold there are right 3 samples in the test set. The classification performance of 20 runs of nine-fold cross-validation is given in Table 4.2.

As shown in Table 4.2, uHONMFtf obtains the highest mean prediction accuracy (0.8148). uHOSVDls, uHOOIls, and uHOOItf obtain similar accuracies. This means that the tensor-decomposition-based unsupervised methods can capture discriminative information. uHONMFtf outperforms the HOSVD and HOOI based methods perhaps due to non-negativity. The reasons why uHONMFls and uHOOItf do not performed well needs further investigation. Good performance is also achieved by the supervised sHOSVD.

From Table 4.2, we can conclude that the multi-dimensional reduction techniques are able to dramatically reduce the dimension of the original tensor data and can transform the

[2] We also tried SVM which gave similar results.

Table 4.2: Classification performance of tensor factorization on complete IFNβ data.

Method	Parameter	Specificity	Sensitivity	Accuracy
uHOSVDls	7,3,3	0.8389±0.039	0.5944±0.020	0.7574±0.050
uHOOIls	4,3,10	0.9000±0.031	0.5000±0.012	0.7667±0.035
uHONMFls	3,5,3	0.8972±0.079	0.3056±0.034	0.7000±0.052
uHOSVDtf	4,2,3	0.7639±0.053	0.5500±0.041	0.6926±0.046
uHOOItf	3,7,3	0.8111±0.048	0.6611±0.055	0.7611±0.050
uHONMFtf	3,5,3	0.7889±0.029	0.8667±0.154	0.8148±0.040
sHOSVD	4,3,8	0.8306±0.054	0.6333±0.012	0.7648±0.044
sHOOI	3,4,4	0.7611±0.045	0.6667±0.000	0.7296±0.039
sHONMF	3,4,6	0.9583±0.110	0.0056±0.069	0.6407±0.075

Table 4.3: Comparison of running times on complete IFNβ data.

Method	DiscHMMs	uHOSVDls	uHOOIls	uHONMFtf
Time (seconds)	2.117×10^3	1.321	1.057	1.662×10^3

sample matrices into new "equivalent" short vectors which are used for classification. In uHONMFtf for example, a 53 by 7 test sample can be represented by a vector of length 3 in the new feature space; thus reducing the data by 99.19% while preserving discriminative information.

The computing times (in seconds) of the tensor-factorization method and a hidden-Markov-model based method are compared in Table 4.3. The number of selected genes is set to 7 for DiscHMMs. The tensor-decomposition-based approaches use the same parameter $(3, 5, 3)$ which are the numbers of factors in the three axes. In Table 4.1, it can be seen that the tensor-factorization-based methods, HOSVD and HOOI, are much faster than the HMM based method while giving at least comparable classification results. uHONMFtf also took less time than DiscHMMs. If other optimization algorithms, such as active-set non-negative least squares algorithm, are used, we believe that the time-complexity of HONMF can be dramatically improved.

Finally, we should remind the reader, who is willing to use NMF (or HONMF) to analyze their microarray data, that the non-negativity of the data must be examined before using it. If the data have negative values, the non-negativity constraint should be enforced only on the coefficient matrix (or mode matrices of HONMF).

3 Kernel Methods for Classification

In the last section, hidden Markov models and tensor factorization techniques are discussed for the classification of GST data. The hidden Markov models make direct use of the temporal information in the GST data, while tensor factorization methods separate the dimension reduction from the classification phases. In this section, we shall show that the similarity between a pair of samples can be measured by a dynamical systems kernel, and many kernel classification and kernel dimensionality reduction methods can be used by taking the kernel matrices (rather than the two-way original samples) as inputs. That is, the corresponding classification and dimensionality reduction is *dimension-free*. Therefore, one may not need to propose new classifiers and dimensionality reduction techniques for GST data. In the following, we first review the existing method using a dynamical systems kernel and the SVM classifier. We then propose to use kernel sparse coding and kernel dictionary learning methods.

3.1 Related Works

The authors of [124] used SVM classifier based on *dynamical systems kernel* to classify GST samples. Since each GST sample is represented by a time-series matrix. It is not appropriate to use the kernels, for example *radial basis functions* (RBF), which take vectorial inputs, because the temporal structure would be deteriorated by vectorization. Dynamical systems kernel accepts matrix inputs and takes the temporal information into account. We define the dynamical systems kernel as follows. Two time-series matrix samples, say $\boldsymbol{X}$ and $\boldsymbol{X}'$, can be modeled by two separate *linear time invariant* (LTI) dynamical systems $\boldsymbol{X} = (\boldsymbol{P}, \boldsymbol{Q}, \boldsymbol{R}, \boldsymbol{S}, \boldsymbol{x}_0)$ (where $\boldsymbol{x}_0$ is a vector, and $\boldsymbol{P}$, $\boldsymbol{Q}$, $\boldsymbol{R}$, and $\boldsymbol{S}$ are matrices estimated by a SVD based approach) and $\boldsymbol{X}' = (\boldsymbol{P}', \boldsymbol{Q}', \boldsymbol{R}', \boldsymbol{S}', \boldsymbol{x}_0')$. The dynamical systems kernel between $\boldsymbol{X}$ and $\boldsymbol{X}'$ is defined as

$$k(\boldsymbol{X}, \boldsymbol{X}') = \boldsymbol{x}_0^{\mathrm{T}} \boldsymbol{M}_1 \boldsymbol{x}_0' + \frac{1}{\mathrm{e}^{\lambda} - 1}[\mathrm{trace}(\boldsymbol{S}\boldsymbol{M}_2) + \mathrm{trace}(\boldsymbol{R})], \tag{4.11}$$

where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ satisfy the Sylvester equation [124], and λ is a positive parameter of the kernel.

We shall investigate SVM in details in Appendix B. In this chapter, for completeness, we briefly introduce SVM on two-way data, and then show that only inner products of samples are needed. The SVM is a basis-expanded linear model which can be formulated as:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x} + b, \tag{4.12}$$

where $\boldsymbol{w}$ is the normal vector to the hyperplane, and b is the bias. The decision function is the indicator:

$$d(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x}|\boldsymbol{w}^*, b^*)], \tag{4.13}$$

with $\{\boldsymbol{w}^*, b^*\}$ being the optimal parameter with respect to some criteria. The geometric interpretation of the standard SVM is that the margin between two classes is maximized while keeping the samples of the same class at one side of the margin.

Suppose a two-way training set is represented by a matrix $\boldsymbol{X} \in \mathbb{R}^{m \times n}$, where each column corresponds to a training sample, and the class labels are in the column vector $\boldsymbol{y} \in \{-1, +1\}^n$. We define $\boldsymbol{Z}$ to be the sign-changed training samples with its i-th column defined as the element-wise multiplication of the class label and the input vector of the i-th training sample, that is, $\boldsymbol{z}_i = y_i * \boldsymbol{x}_i$. The optimization of the soft-margin C-SVM can be formulated as:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ &\text{s.t. } \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \boldsymbol{1} - \boldsymbol{\xi} \\ &\qquad \boldsymbol{\xi} \geq 0, \end{aligned} \tag{4.14}$$

where $\boldsymbol{C} = \{C\}^m$ controls the tradeoff between the regularization term and the loss term, and $\boldsymbol{\xi}$ is a column vector of slack variables.

By considering corresponding Lagrange function and *Karush-Kuhn-Tucker* (KKT) conditions, we can obtain the dual form of the optimization:

$$\begin{aligned} &\min_{\boldsymbol{\mu}} \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{1} \\ &\text{s.t. } \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ &\qquad 0 \leq \boldsymbol{\mu} \leq \boldsymbol{C}, \end{aligned} \tag{4.15}$$

where $\boldsymbol{\mu}$ is the vector of Lagrange multipliers ($\boldsymbol{\mu}$ is a sparse vector). The nonzero multipliers correspond to the *support vectors*, which are crucial for classification. The training samples corresponding to zero multipliers can be ignored in further computations. The relation between the dual variable $\boldsymbol{\mu}$ and the primal variable $\boldsymbol{w}$ is $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{\mu} * \boldsymbol{y}) = \boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)$, where S is the set of indices of the non-zero multipliers. The bias b can be computed by $b = \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{w}}{|B|} = \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)}{|B|}$, where B is the index of the nonzero and unbounded multipliers corresponding to the support vectors on the margin border. From the formulation

Table 4.4: Comparison of classification performance of SVM on complete IFNβ data.

Method	Parameter	Specificity	Sensitivity	Accuracy	Time
rbfSVM	1	1.000±0.000	0.000±0.000	0.667±0.000	-
dsSVM	1,5	0.972±0.082	0.422±0.013	0.789±0.023	93.474

of the optimal $\boldsymbol{w}^*$ and b^*, the linear function in the decision function can be computed as:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{*\mathrm{T}}\boldsymbol{x} + b^* = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S^* * \boldsymbol{y}_S) + \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S^* * \boldsymbol{y}_S)}{|B|}. \tag{4.16}$$

From Equations (4.15) and (4.16), we can see that the optimization and decision making of SVM only need the inner products of the training samples. By replacing the inner products by appropriate kernel functions, we can classify any data. Therefore, SVM with kernels is dimension-free. By using dynamical systems kernel, we can use SVM to classify time-series matrix samples. We compared the performance of SVM using dynamical systems kernel with that using RBF kernel on IFNβ data. Both methods are denoted by dsSVM and rbfSVM, respectively. The same experimental setting as in the previous section is used here. The comparison of both methods is given in Table 4.4. The parameter of rbfSVM is the parameter of the RBF function. The first parameter of dsSVM is the number of hidden states, and the second one is the parameter of the dynamical systems kernel function. We can see that rbfSVM fails to identify any positive sample, while dsSVM obtains a sensitivity of 0.422. The overall accuracy of dsSVM is 0.789, which is much higher than rbfSVM. Thus, we can conclude that the classification performance can be improved by considering structural information in GST data. Furthermore, the SVM method is more efficient than the hidden Markov models (as shown in Table 4.3).

3.2 Kernel Sparse Coding Method for Classification

We have already introduced sparse-coding based classification techniques in Chapter 2. Now, we extend the NNLS sparse coding method for tensor data. Without loss of generality, we suppose there are I_3 training samples represented by a three-way tensor $\boldsymbol{\mathcal{A}}_{I_1\times I_2\times I_3}$. The third axis is for the samples. Each sample is a matrix of size $I_1 \times I_2$. Therefore, $\boldsymbol{\mathcal{A}}(:,:,i)$ is the i-th training sample. Suppose we have P new samples in $\boldsymbol{\mathcal{B}}_{I_1\times I_2\times P}$. Assume that each of such new samples can be regressed by a non-negative linear combination of the training

Table 4.5: Comparison of classification performance of sparse coding on complete IFNβ data.

Method	Parameter	Specificity	Sensitivity	Accuracy	Time
l_1LS	2^{-4}	0.9000±0.0329	0.3750±0	0.7444±0.0231	0.1152
NNLS	-	0.9000±0.0329	0.3750±0	0.7444±0.0231	0.0573
Kl_1LS	2^{-14},[1,5]	0.7316±0.0468	0.7937±0.0596	0.7500±0.0386	4.2666
KNNLS	[1,5]	0.7895±0.0408	0.7375±0.0375	0.7741±0.0308	4.5199

samples. We then need to solve the following NNLS problem:

$$\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{\mathcal{B}} - \boldsymbol{\mathcal{A}} \times_3 \boldsymbol{X}^{\mathrm{T}}\|_F^2, \text{ s.t. } \boldsymbol{X} \geq 0, \tag{4.17}$$

where $\times_3$ is the mode-3 product [125] as defined in Appendix A. Through matricizing tensors to matrices, we can convert the above optimization task into the equivalent formula:

$$\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{B}_{(3)} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{A}_{(3)}\|_F^2, \text{ s.t. } \boldsymbol{X} \geq 0, \tag{4.18}$$

where $\boldsymbol{A}_{(3)}$ is a matrix of size $I_3 \times (I_1 \times I_2)$, unfolded from tensor $\boldsymbol{\mathcal{A}}$ in mode 3. Using transposition, we have

$$\min_{\boldsymbol{X}} \frac{1}{2}\|\boldsymbol{B}_{(3)}^{\mathrm{T}} - \boldsymbol{A}_{(3)}^{\mathrm{T}}\boldsymbol{X}\|_F^2, \text{ s.t. } \boldsymbol{X} \geq 0. \tag{4.19}$$

Now, Equation (4.19) can be solved by a two-way non-negative least squares algorithm, for example, the FC-NNLS in [3]. Our NNLS classifier can now be generalized for tensor data.

However, the drawback of this generalization is that the structural information within a sample is not considered. The objective (Equation (4.17)) uses the Euclidean distance, hence the samples are actually vectorized in Equations (4.18) and (4.19). If we use other dissimilarity or similarity metrics which take the temporal information into account when classifying gene-sample-time data, the performance is expected to be increased. Since dynamical systems kernel, defined in Equation (4.11), accepts matrix inputs and takes the temporal information into account, we can thus apply it to our kernel NNLS method for GST data.

Using the same experimental setting as in previous sections, the classification performances of linear l_1LS and NNLS and their kernel versions on GST data are compared in Table 4.5. We have the following observations. First of all, from the comparison between

Table 4.6: Comparison of classification performance of kernel dictionary learning on complete IFNβ data.

Method	Prior	Specificity	Sensitivity	Accuracy	Time
KDL-l_1LS	Gaussian	0.9711±0.0352	0.1250±0.1118	0.7204±0.0320	39.9439
KDL-NNLS	Gaussian	0.9711±0.0310	0.1081±0	0.7370±0.0329	13.3402
KDL-l_1NNLS	Gaussian	0.9579±0.0394	0.1437±0.0501	0.7167±0.0724	14.9027
KDL-l_1LS	uniform	0.9684±0.0349	0.1375±0.1111	0.7222±0.0321	29.7428
KDL-NNLS	uniform	0.9711±0.0310	0.1437±0.1066	0.7259±0.0429	7.4780
KDL-l_1NNLS	uniform	0.9763±0.0310	0.1563±0.1178	0.7333±0.0505	13.7721

NNLS using linear kernel and its counterpart using dynamical-systems kernel, we can see that the latter obtained better results, because it considers temporal information within the samples. Thus, the structural information within the samples does contribute to the discrimination. Second, through comparing the computing time (in seconds), we can see that the sparse coding methods are very fast, compared with the hidden Markov models, some tensor decomposition methods and SVM (as shown in Tables 4.3 and 4.4).

3.3 Kernel NMF and Dictionary Learning

NMF has been applied to dimensionality reduction for gene-sample data in [43]. Also, we have shown, in Section 2, that the high-order NMF can be applied on GST data. In this section, we shall show that feature extraction can be conducted directly by kernel NMF and kernel dictionary learning in sparse representation.

In Chapter 2, we have derived the dimensionality reduction method based on kernel dictionary learning. Using a suitable kernel to measure the similarity between a pair of time-series samples, kernel dictionary learning can be applied on GST data for dimensionality reduction. A computational experiment is given in the following. We used the IFNβ data again. We used dynamical systems kernel with the same parameters as in previous sections. SVM was employed after dimensionality reduction. The experimental results are shown in Table 4.6. We can see that the kernel sparse representation methods obtained lower sensitivity compared with the results of sparse coding methods in Table 4.5. This may be because the number of positive samples in the training set is very small, around eight, which may be insufficient for some dimensionality reduction techniques. However, it would be very interesting to investigate the performance of kernel sparse representation on a larger data set due to its computational flexibility. Finally, we can see that the kernel dictionary learning methods are very efficient for GST data as well.

4 Conclusions

In this chapter, we discussed different machine learning approaches for the classification and dimensionality reduction of GST data. We proposed the feature extraction method based on dense and sparse tensor decompositions. We also used kernel sparse coding and kernel dictionary learning techniques for classifying GST data given an appropriate kernel, which can measure similarity between a pair of time-series samples.

There are still many challenges in each category. The current HONMF is implemented by multiplicative update rules. Using other optimization techniques, for example active-set algorithms as in NMF, may dramatically improve the efficiency. The current dynamical systems kernel is not symmetric and is unbounded, and hence a better kernel is needed to overcome these shortcomings. The most popular data set in this area is IFNβ which has a small numbers of genes, samples, and time points. More GST data sets (if exist) need to be obtained from the *Gene Expression Omnibus* [126] in order to investigate the performance of the methods mentioned in this chapter.

Publications

1. The sparse tensor decomposition method for the dimensionality reduction of GST data were first published in [*Y. Li and A. Ngom, "Classification of clinical gene-sample-time microarray expression data via tensor decomposition methods," LNBI/LNCS: Selected Papers of 2010 International Meeting on Computational Intelligence Methods for Bioinfomatics and Biostatistics (CIBB), vol. 6685, pp. 275-286, 2011.*].

2. We applied kernel sparse coding and kernel dictionary learning methods for classifying matrix samples and extract vectorial features, respectively, in the book chapter [*Y. Li and A. Ngom, "Mining gene-sample-time microarray data," in L. Rueda ed. Microarray Image and Data Analysis: Theory and Practice, CRC Press/Taylor & Francis, Dec. 2013, in press.*].

Chapter 5

Sparse Regularized Linear Models for Multi-class Classification

1 Introduction

Regularized linear models are linear models which minimize the empirical error and model complexity simultaneously. The regularized linear models are usually sparse. We say a regularized model is *sparse*, if either the model parameter is a sparse linear combination of training samples, or the model parameter is sparse. *Support vector machines* (SVMs) are state-of-the-art models in this class. The l_2-norm SVMs fall in the the former case, and the l_1-norm SVM falls into the latter case. In addition to classification, the sparse regularized linear models can be applied to sample selection and feature selection in virtue of the sparsity. We refer the readers to Appendix B for a detailed review of regularized linear models including model formulations, optimizations, and machine learning applications.

They have been widely used in the analysis of high-dimensional biological data. For example, SVMs have been applied in classifying tissue samples using microarray expression data [4]. In [127], the *SVM-based recursive feature elimination* (SVM-RFE) was proposed to select discriminative genes for cancer classification. In [128], SVM was applied as a sample selection method for gene selection. The l_1-norm SVM can simultaneously select variables and classify instances. It has been used for gene selection [129] as well.

One weakness of a linear model is that, it separates the feature space into only two open regions, thus it can not be *naturally* extended to the multi-class case. There are some strategies that extend linear models to classify multi-class data. Among them, the *one-against-all* and *one-against-one* are two well-known ones.

Contributions: In this chapter, we propose two novel strategies to extend linear models to multi-class data. The first one is an independent modeling, called nearest border method. The second one is a hierarchical model. The contributions of this chapter include:

1. We propose the novel nearest border paradigm for multi-class classification problems, and implemented it by using a one-class SVM. This philosophy has not been presented before in the literature.

2. We propose the hierarchical model for multi-class classification problems. This model is so flexible that any feature selection and classification methods can be embedded in the model. We apply this model for gene selection and prediction of breast tumor subtypes, simultaneously.

3. We realize most of the regularized linear model based methods mentioned in this chapter and Appendix B in MATLAB, and assemble them to an open-source toolbox named *regularized linear models and kernels* (RLMK) toolbox.

In the following, we shall first present the nearest border method in Section 2. Then, we propose the hierarchical model in Section 3. Finally, we introduce our implementations of these models and many regularized linear models in Section 4.

2 Nearest Border Technique [1]

2.1 Introduction

Overview and Related Fields

One goal of this section is to present a new paradigm in pattern recognition (PR) , which we shall refer to as the *nearest border* (NB) paradigm. This archetype possesses similarities to many of the well-established methodologies in PR, and can also be seen to include many of *their* salient facets/traits. In order for the reader to capture the intricacies of our contribution, and be able to perceive it in the context of the existing state of the art, in this introductory section, we briefly describe some of these methodologies from a *conceptual* perspective.

The problem of classification in machine learning can be quite simply described as follows: If we are given a limited number of training samples, and if the class-conditional distributions are unknown, the task at hand is to predict the class label of a new sample with

[1] This section is based on our collaborative work accepted as [130], with Dr. B. John Oommen of University of Carleton, Ottawa.

minimum risk. Within the generative model of computation, one resorts to modeling the distribution of the prior, and then computing the *a posteriori* distribution after the testing sample arrives. The strength of this strategy is that one obtains an optimal performance if the assumed distribution is the same as the actual one. The limitation, of course, is that it is often difficult, if not impossible, to compute the posterior distribution. The alternative is to work with methods that directly model the latter posterior distribution itself. These methods differ in the approximation of the posterior, such as the *nearest neighbor* (NN) or the *k-nearest neighbors* (k-NN), the *support vector machine* (SVM) etc. This section advocates such a philosophy.

The most common challenges that all these techniques encounter are (i) the *curse of dimensionality*, which is encountered when the dimensionality of the feature space is large, (ii) the *small sample size* scenario encountered when one attempts to obtain a significant performance even though the size of the training set is small, (iii) the *large sample size* scenario, in which the computational resources used are large because of the high cardinality of the training set.

For decades, the NN or k-NN classifiers have been widely-used classification rules. Each class is described using a set of sample prototypes, and the class-identity of an unknown vector is decided based on the identity of the closest neighbor(s), which are found among all the prototypes [131]. This rule is simple, and yet it is one of the most efficient classification rules in practice. The application of the classifier, however, often suffers from the higher order of the computational complexity caused by the large number of distance computations, especially as the size of the training set increases in high dimensional problems [132], [131]. Strategies that have been proposed to solve this dilemma can be summarized into the following categories: (i) reducing the size of the design set without sacrificing the performance, (ii) accelerating the computation by eliminating the necessity of calculating superfluous distances, and (iii) increasing the accuracy of the classifiers designed with the set of limited samples.

A simple strategy for affecting this is, for example, that of: (i) using the mean of the training samples of a class in nearest centroid-like method, (ii) resorting to *vector quantization* (VQ), and (iii) invoking the *non-negative matrix factorization* (NMF) scheme, among others. The strengths of these are that the accuracy may not deteriorate by using only a fewer number of samples or meta-samples, and this can be useful when the data is noisy and/or redundant. One must observe that the testing algorithm is, by definition, faster. The weakness of using a simple parametric strategy, (e.g., the nearest centroid scheme) is that the sample mean is merely the first order moment, and does not consider

higher order statistics.

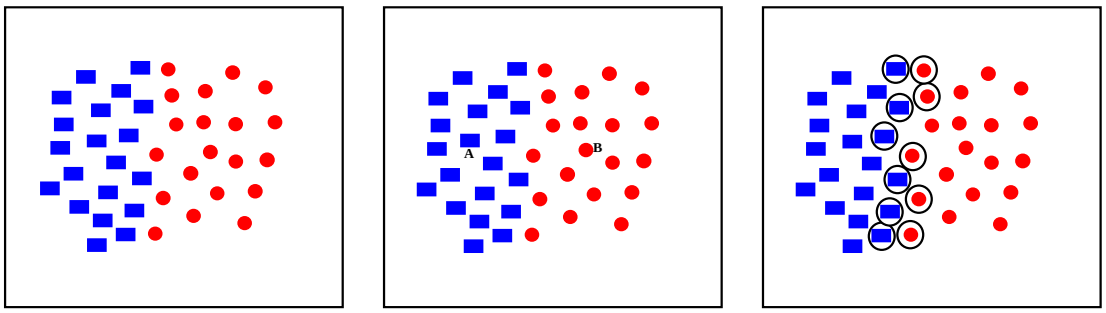
The four families of algorithms, which are most closely related to the NB paradigm that we propose, are briefly surveyed below.

Prototype Reduction Schemes: The first of the solutions mentioned above, i.e., of reducing the size of the design set without sacrificing the performance, is the basis for the family of *prototype reduction schemes* (PRSs), which is central to this section. The goal here is to reduce the number of training vectors while simultaneously insisting that the classifiers built on the reduced design set perform as well, or nearly as well, as the classifiers built on the original design set. Thus, instead of considering all the training patterns for the classification, a subset of the whole set is selected based on certain criteria. The learning (or training) is then performed on this reduced training set, also called the *reference* set. This idea has been explored for various purposes, and has resulted in the development of many algorithms surveyed in [133, 134, 135]. It is interesting to note that Bezdek *et al.* [133], who have composed an excellent survey of the field, report that there are "zillions!" of methods[2] for finding prototypes (see page 1,459 of [133]). There are also many *families* of PRSs. In certain families, Not only does this reference set contains the patterns which are closer to the true discriminant's boundary, but also the patterns from the other regions of the space that can adequately represent the entire training set.

Border Identification Algorithms: *border identification* (BI) algorithms, which are a subset of PRSs, work with a reference set that contains only "border" points. To enable the reader to perceive the difference between general PRSs and BI algorithms, we present some typical data points in Figure 5.1. Consider Figure 5.1a in which the circles belong to class ω_1 and rectangles belong to class ω_2. A PRS would attempt to determine the relevant samples in both the classes which are capable of achieving near-optimal classification. Observe that some samples which fall strictly *within* the collection of points in each class, such as A and B in Figure 5.1b, could be *prototypes*, because the testing samples that fall close to them will be correctly classified. As opposed to this, in a BI algorithm, one uses *only* those samples that lie close to the *boundaries* of the two classes, as shown in Figure 5.1c. In all brevity, we mention that recent research [146] has shown that for overseeing the task of

[2]One of the first of its kind is the condensed nearest neighbor (CNN) rule [136]. The CNN, however, includes "interior" samples which can be eliminated completely without changes in the performance. Accordingly, other methods have been proposed successively, such as the reduced nearest neighbor (RNN) rule [137], the prototypes for nearest neighbor (PNN) classifiers [138], the selective nearest neighbor (SNN) rule [139], two modifications of the CNN [140], the edited nearest neighbor (ENN) rule [141], and the non-parametric data reduction method [142]. Additionally, in [143], the vector quantization (VQ) technique [144] was also reported as an extremely effective approach to data reduction. It has also been shown that the SVM can be used as a mean of selecting initial prototype vectors, which are subsequently operated on by LVQ3-type methods [145].

achieving the classification, the samples extracted by a BI scheme, and which lie *close to the discriminant function's boundaries*, have significant information when it concerns the power of the classifier. Duch [147] and Foody [148] proposed algorithms to achieve this. But as the patterns of the reference set described in [147] and [148] are only the "near" borders, they do not have the potential to represent the entire training set, and hence do not perform well. In order to compete with other classification strategies, it has been shown that we need to also include the set of "far" borders to the reference set [146]. A detailed description of traditional BI algorithms namely Duch's approach, Foody's algorithm and the *border identification in two stages* can be found in [149].



(a) Training set. (b) Prototypes. (c) Border patterns.

Figure 5.1: A schematic view which shows the difference between *border* patterns and *prototypes*.

SVM-type Algorithms: A representative of a completely distinct family of algorithms is the acclaimed SVM which is known as being quite suitable from a theoretical point of view as well as in practical applications. From the basic theory of the SVM (explained later) we know that it has the capability of extracting vectors which support the boundary between the two classes, and they can satisfactorily represent the global distribution structure. Also the learning algorithm can be easily expanded to nonlinear problems by employing a technique akin to that of kernel functions. As we shall demonstrate in a subsequent section, our NB paradigm is actually very closely linked to the family of SVM schemes. Readers are referred to Appendix B for a review of SVM and many other regularized linear models.

"Anti-Bayesian" OS-Based Algorithms: A relatively new and distinct paradigm, which works in a counter-intuitive manner, is the recently introduced "anti-Bayesian" philosophy. As a back-drop to this, we mention that when expressions for the *a posteriori* distribution are simplified, the classification criterion that attains the Bayesian optimal lower bound often reduces to testing the sample point using the corresponding distances/norms

to the *means* or the "central points" of the distributions. In [150, 151, 149], the authors demonstrated that they can obtain optimal results by operating in a diametrically opposite way, i.e., a so-called "anti-Bayesian" manner. They showed that by working with a *few* points *distant* from the mean, one can obtain remarkable classification accuracies. The number of points referred to can be as small as *two* in the uni-dimensional case. Further, if these points are determined by the *order statistics* (OS) of the distributions, the accuracy attains the optimal Bayes' bound. They demonstrated that one could work with the OS of the features rather than the distributions of the features themselves, and proposed the strategy referred to as *classification by moments of order statistics* (CMOS) [150, 151, 149]. It turns out, though, that this process is computationally not any more complex than working with the latter distributions.

The state of the art of OS-based classification is summarized below. Initially, in [149], the authors worked with the OS for the data points, and showed how it could achieve near-optimal classification for various uni-dimensional distributions. For uni-dimensional OS-based PR, their methodology is based on considering the n-order OSs, and comparing the testing sample with the $n-k^{th}$ OS of the first distribution and the k^{th} OS of the second. Equivalently, these points are determined by the location of the $(\frac{n-k+1}{n+1})^{th}$ percentile of the first distribution and the $(\frac{k}{n+1})^{th}$ percentile of the second distribution. These results were shown to be applicable for the distributions that are members of the symmetric exponential family. By considering the entire spectrum of the possible values of k, the results in [150] and [149] showed that the specific value of k is usually not so crucial. Subsequently, in [151], they proved that these results can also be extended for multi-dimensional distributions.

The challenge involved in using OS criteria is that one needs many training samples to estimate the OS. Thus, the question of resolving this for the small sample set is still open.

This brings us to the question of why one needs a new paradigm and what this paradigm entails.

2.2 Problem Formulation

In this section, we would like to explicitly formulate a paradigm for PR that only uses the "border" points. First of all, the goal is that this process should be independent of the number of dimensions, thus overcoming a handicap inherent in the OS-based schemes. This would, thus, permit us to apply the BI principle to high-dimensional data. The method that we propose should encapsulate a methodology that is universal for any distribution and should, hopefully, simultaneously crystallize the concept of the border in the multivariate case.

What then does this new paradigm entail? Essentially, we would like it to possess all the salient characteristics of all the four families of methods described above. First and foremost, it should be able to learn the border for each class. To achieve this, unlike the traditional BI methods, we shall not resort to using *inter*-class criteria. Rather, we shall compute the border for a specific class in the d-dimensional hyper-space by invoking *only* the properties of the samples *within that class*. Once these borders have been obtained, we advocate that testing is accomplished by assigning the test sample to the class whose border it lies closest to. We claim that this distance is an approximation to the value of the *a posteriori* distribution, which justifies the rule of assigning the testing samples to the nearest border. This claim, is actually counter-intuitive, because unlike the centroid or the median, these border samples are often "outliers" and are, indeed, the points that represent the class the least.

Proposed Solution: The heart of our solution is that we apply the *support vector domain description* (SVDD) for the multi-class problems for which the authors of [152] earlier proposed a Bayesian method. First of all, we learn a SVDD representation for each class. Thereafter, we construct a pseudo-class-conditional-density function for each class. Finally, the decision is made using the estimated pseudo-posterior probabilities. In this regard, the authors of [153] proposed a multi-class classifier by an ensemble of one-class classifiers. First of all, a SVDD or *kernel principal component analysis-based kernel whitening* (KW-KPCA) is applied to each class, where we can see that the SVDD approximates the class boundary by hyper-spheres in the feature space, while the KW-KPCA uses hyper-ellipses. Thereafter, the normalized distance from the prototype of each class is computed, whence the testing sample is assigned to the class which minimizes this distance.

The novel contributions of this section are the following:

- We propose a new PR paradigm, the nearest border paradigm, in which we create borders for each individual class, and where testing is accomplished by assigning the test sample to the class whose border it lies closest to.
- Our paradigm falls within the family of PRSs, because it yields a reference set which is a small subset of original training patterns. The testing is achieved by *only* utilizing the latter.
- Our paradigm falls within the family of BI methods, except that unlike traditional BI methods, the borders we obtain do not use *inter*-class criteria; rather, they *only* utilize the properties of the samples *within that class*.

- The nearest border paradigm is essentially "anti-Bayesian" in its salient characteristics. This is because the testing is not done based on central concepts such as the centroid or the median, but by comparisons using these border samples, which are often "outliers" and which, in one sense, represent the class the least.
- The nearest border paradigm is closely related to the family of SVMs, because the computations and optimization used are similar to those involved in deriving SVMs.
- To justify all these claims, we submit a formal analysis and the results of various experiments which have been performed for many distributions and for many real-life data sets, and the results are clearly conclusive.

We conclude by mentioning that, as far as we know, such a paradigm has not been reported in the PR literature.

The rest of the section is organized as follows. First of all, in Section 2.3, we present an overall overview of the NB philosophy. In Section 2.4, we implement this philosophy by using SVDD. Section 2.5 details the experimental results obtained by testing our schemes and comparing it with a set of benchmark algorithms.

In the next subsection, we shall formalize the general theory of the NB classification paradigm.

2.3 Theory of The Nearest Border (NB) Paradigm

We assume that we are dealing with a pattern recognition problem involving g classes: $\{\omega_1, \cdots, \omega_g\}$. For any specific class ω_i, we define a region $\mathcal{R}_i$ that is described by the function $f_i(\boldsymbol{x}) = 0$ (which we shall refer to as its "border"), where $\mathcal{R}_i = \{\boldsymbol{x} | f_i(\boldsymbol{x}) > 0\}$. We describe $\mathcal{R}_i$ in this manner so that it is able to capture the main mass of the probability distribution $p_i(\boldsymbol{x}) = p(\boldsymbol{x}|\omega_i)$. All points that lie outside of $\mathcal{R}_i$, are said to fall in its "outer" region, $\bar{\mathcal{R}}_i$, where $\bar{\mathcal{R}}_i = \{\boldsymbol{x} | f_i(\boldsymbol{x}) < 0\}$. These points are treated as outliers as far as class ω_i is concerned.

The function $f_i(\boldsymbol{x})$ is crucial to our technique, because it explicitly defines the region $\mathcal{R}_i$. Formally, the function $f_i(\boldsymbol{x})$ must be defined in such a way that:

1. $f_i(\boldsymbol{x})$ is the *signed distance* from the point $\boldsymbol{x}$ to the border such that $f_i(\boldsymbol{x}) > 0$ if $\boldsymbol{x} \in \mathcal{R}_i$, and $f_i(\boldsymbol{x}) < 0$ if $\boldsymbol{x} \in \bar{\mathcal{R}}_i$;
2. If $f_i(\boldsymbol{x}_1) > f_i(\boldsymbol{x}_2)$, then $p_i(\boldsymbol{x}_1) > p_i(\boldsymbol{x}_2)$;
3. If $f_i(\boldsymbol{x}) > f_j(\boldsymbol{x})$, then $p(w_i|\boldsymbol{x}) > p(w_j|\boldsymbol{x})$.

In order to predict the class label of a new testing sample $\boldsymbol{x}$, we calculate its signed distance from each class, and thereafter assign it to the class with the minimum distance. In other words, we invoke the softmax rule: $j = \arg\max_{i=1}^{g} f_i(\boldsymbol{x})$.

The main challenge that we face in formulating, designing and implementing such a NB theory lies in the complexity of conveniently and accurately procuring such borders. The reader will easily see that this is equivalent to the problem of identifying functions $\{f_i(\boldsymbol{x})\}$ that satisfy the above constraints. Although a host of methods to do this are possible, in the following, we propose one that identifies the boundaries using the one-class SVM.

2.4 NB Classifiers: The Implementations of The NB Paradigm

Before reading the following content, we refer the readers to the Appendix B for an introduction to one-class SVMs and ν-SVM. We are now discuss how the one-class SVMs can be used to formulate the family of NB classifiers. To do this, we shall first affirm that the two-class SVM actually consists of two hyperplane-based one-class SVMs. Thereafter, we shall present the implementation of the NB paradigm based on the hypersphere-based SVDD.

One-Class SVM-Based Schemes

We shall first state the relationship between the family of hyperplane-based one-class SVMs and the corresponding two-class SVM. This result is given by the following proposition.

Proposition 2. *For two-class data, the task of learning a single two-class SVM is equivalent to that of learning two one-class SVMs under the condition that the hyperplanes of both the one-class SVMs are parallel.*

Proof. Without loss of generality, in our proof, we shall assume that we are considering the case of obtaining the two-class ν-SVM. Let us suppose that the parallel hyperplanes for the positive and negative classes are: $f_+(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} - b_+ = 0$, and $f_-(\boldsymbol{x}) = (-\boldsymbol{w})^{\mathrm{T}}\boldsymbol{x} + b_- = 0$, where the biases $b_+, b_- > 0$. With regard to the signs of the respective functions, we mention that:

1. $f_+(\boldsymbol{x}) > 0$ if $\boldsymbol{x}$ is on the positive side (the side $\boldsymbol{w}$ pointing to) of $f_+(\boldsymbol{x}) = 0$.
2. $f_-(\boldsymbol{x}) > 0$ if $\boldsymbol{x}$ is on the positive side (the side $-\boldsymbol{w}$ pointing to) of $f_-(\boldsymbol{x}) = 0$.

Now consider the optimization associated with learning of two one-class SVMs with

parallel hyperplanes. One can see that this can be formulated as below:

$$\begin{aligned} \min_{\boldsymbol{w},b_+,b_-,\boldsymbol{\xi}_+,\boldsymbol{\xi}_-} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{\xi}_+ + \boldsymbol{\xi}_-) - \nu\frac{b_+ + b_-}{2} \\ \text{s.t. } & \phi(\boldsymbol{X}_+)^{\mathrm{T}}\boldsymbol{w} - b_+\mathbf{1} + \boldsymbol{\xi}_+ \geq 0 \\ & -\phi(\boldsymbol{X}_-)^{\mathrm{T}}\boldsymbol{w} - b_-\mathbf{1} + \boldsymbol{\xi}_- \geq 0 \\ & \boldsymbol{\xi}_+, \boldsymbol{\xi}_- \geq 0;\ b_+, b_- > 0 \end{aligned} \tag{5.1}$$

After obtaining the parameters of the model, the hyperplane between the two parallel hyperplanes is $f(x) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \frac{-b_1+b_2}{2} = 0$. If we now re-visit the formulation of the two-class ν-SVM formulation (as given in the Section 2.4 of Appendix B), we see that this is:

$$\begin{aligned} \min_{\boldsymbol{w},b,\rho,\boldsymbol{\xi}} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 - \nu\rho + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t. } & \phi(\boldsymbol{X}_+)^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} - \rho\mathbf{1} + \boldsymbol{\xi}_+ \geq 0 \\ & -\phi(\boldsymbol{X}_-)^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} - \rho\mathbf{1} + \boldsymbol{\xi}_- \geq 0 \\ & \boldsymbol{\xi}_+, \boldsymbol{\xi}_- \geq 0; \rho \geq 0. \end{aligned} \tag{5.2}$$

By a careful examination of the two formulations, one can confirm that we can obtain the *exact same* formulation as in Equation (5.1) by setting $b = \frac{-b_+ + b_-}{2}$ and $\rho = \frac{b_+ + b_-}{2}$, where $b_+ > 0$ and $b_- > 0$. This concludes the proof. □

Remark: From the above proposition, we can further infer that the two-class SVM is, in fact, an implementation of what we have referred to as the *nearest border* paradigm! This is because, whenever we want to assign a new sample, $\boldsymbol{x}$, to a specific class, the SVM decision function: $d(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})]$, is equivalent to:

$$j = \arg\max_{i=+/-} f_i(\boldsymbol{x}) = \arg\max_{i=+/-} \frac{f_i(\boldsymbol{x})}{\|\boldsymbol{w}\|}. \tag{5.3}$$

Further, from the Bayesian learning theory, this formulation is precisely a discriminative model that directly models the *a posteriori* probability distribution.

The Hypersphere-Based *Nearest Border* Method

The nearest centroid approach only uses the means of the class-conditional distribution, and this is the reason why it is not effective for the scenario when the variances of the various classes are very different. As shown above, the two-class SVM can find the boundary of

each class, but the solution to this problem cannot be easily and naturally extended to the multi-class problem. The difficulty of extending any linear model from its two-class formulation to its corresponding multi-class formulation, lies in the fact that a hyperplane always partitions the feature space into two "open" subspaces, implying that this can lead to ambiguous regions that may be generated by some extensions of the two-class regions for the multi-class case. The most popular schemes to resolve this are the one-against-rest (using a softmax function) and the one-against-one solutions.

Viewed as a one-class model, the work based on Tax and Duin's SVDD [154] aims to find a closed hypersphere in the feature space that captures the main part of the distribution. By examining the corresponding SVM, we see that the hypersphere obtained by the SVDD is the estimate of the features' *highest density region* (HDR). In particular, for the univariate distribution, the estimation of the *highest density interval* (HDI) involves searching for the threshold p^* that satisfies:

$$\int_{x:p(x|\boldsymbol{D})>p^*} p(x|\boldsymbol{D})dx = 1-\alpha. \tag{5.4}$$

The $(1-\alpha)\%$ HDI is defined as $C_\alpha(p^*) = \{x : p(x|\boldsymbol{D}) \geq p^*\}$. If we now define the *central interval* (CI) by the interval:

$$C_\alpha(l,u) = \{x \in (l,u) | p(l \leq x \leq u|\boldsymbol{D}) = 1-\alpha, p(x \leq l) = \frac{\alpha}{2}, p(x \geq u) = \frac{\alpha}{2}\},$$

one will see that, for symmetric unimodal univariate distribution, HDI coincides with the CI. However, for nonsymmetric univariate distributions, the HDI is smaller than the CI.

For known distributions, the CI can be estimated by the corresponding quantile. However, for unknown distributions, the CI can be estimated by a Monte Carlo approximation (or by the histogram, or the *order statistics* [149]). However, in the context of this study, we remark that by virtue of Vapnik's principle, it is not necessary to estimate the density by invoking a non-parametric method.

For multivariate distributions, we can estimate the $(1-\alpha)\%$ HDR $C_\alpha(f)$ by using the equation:

$$\min_f \int_{f(\boldsymbol{x})\geq 0} 1 d\boldsymbol{x}, \text{ s.t. } \int_{\boldsymbol{x}:f(bmx)\geq 0} p(\boldsymbol{x}|\boldsymbol{D})d\boldsymbol{x} = 1-\alpha. \tag{5.5}$$

We shall refer to this optimal contour $f^*(\boldsymbol{x}) = 0$ as the $(1-\alpha)$-border/contour.

Our idea for classification is the following: We can learn a hypersphere for each class in the feature space in order to describe the border of this class. We then calculate the

distance from a unknown sample to the border of each class and assign it to the class with the minimum distance. The training phase of our approach is to learn the hypersphere $f_i(\boldsymbol{x}) = 0$ parameterized by $(\boldsymbol{c}_i, R_i)$ for each class as specified by Equation (B.85) in Appendix B. The prediction phase then involves assigning the unknown sample $\boldsymbol{x}$ using the following rule:

$$j = \arg \max_{i=1}^{g} f_i(\boldsymbol{x}), \tag{5.6}$$

where $f_i(\boldsymbol{x})$ is defined as in Equation (B.91) in Appendix B. In particular, we note that:

1. $f_i(\boldsymbol{x}) \in \mathbb{R}$ is the signed distance of $\boldsymbol{x}$ from the corresponding boundary;
2. For points inside the i-th hypersphere, $f_i(\boldsymbol{x}) > 0$;
3. For points outside the hypersphere, $f_i(\boldsymbol{x}) < 0$. Further, the larger $f_i(\boldsymbol{x})$ is, the closer it is to class ω_i, and the higher the value of $p(w_i|\boldsymbol{x})$ is. From the parameters of $f_i(\boldsymbol{x})$, we can see that $f_i(\boldsymbol{x})$ considers both mean and variance of the distribution. It can be further enhanced by the *normalized distance* through the operation of dividing it by R_i.

This, quite simply, leads us to the following decision rule:

$$j = \arg \max_{i=1}^{g} \frac{f_i(\boldsymbol{x})}{R_i}. \tag{5.7}$$

We refer to this approach as the *nearest border approach based on hypersphere* (NB-HS).

In an analogous manner, the two-class SVM can also be called the *nearest border approach based on hyperplane* (NB-HP). The advantage of using the (normalized) distance from the border instead of the mean as in nearest centroid approach is that the former takes into account both the means and the variances, while the latter considers only the mean. The advantage of the NB-HS over the SVM is that, due to the closure property of the hypersphere, the borders obtained in the NB-HS can be estimated one by one, which is more computationally efficient than by invoking a one-against-all SVM. Hereafter, the hypersphere based NB using the decision rule specified by Equation (5.6) will be denoted by *ν-NB*, and the one that utilizes the normalized distance, as in Equation (5.7) will be denoted by *ν-NBN*.

As mentioned in Section 2.4 of Appendix B, ν is the upper bound of the fraction of outliers and the lower bound of the fraction of the support vectors. As the number of training samples increases to infinity, these two bounds converge to ν. However, in practice, we usually have a very limited number of training samples. In order to obtain ν which

corresponds to the α fraction of outliers, firstly, we need to let $\nu = \alpha$, and then reduce ν gradually until the α fraction of outliers are obtained. This variant of NB will be named the α-*NB* in the subsequent sections.

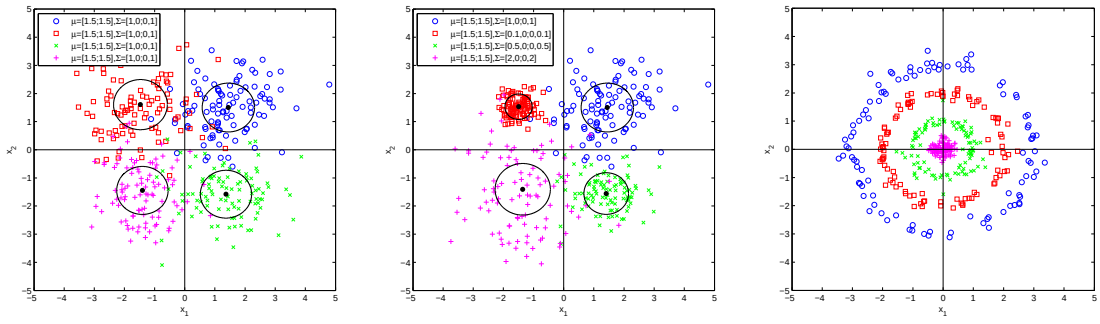
2.5 Experiments

The NB schemes that we introduce in this paper have been rigorously tested. In this section, we present a summary of the experiments done and the corresponding results. Our computational experiments can be divided into two segments. First of all, we investigated the performance of our method on three artificial data sets. Subsequently, we statistically compared our approach with benchmark classifiers on 17 well-known real-life data sets. The methods that we have used and the benchmark methods are listed in Table 5.1.

Before we explain the experimental results we would like to emphasize the fact that we are not attempting to demonstrate that our new technique is the "best available" scheme. Rather, our intention is to show that such a NB strategy is not only feasible – it is also extremely competitive, yielding an accuracy which is close to the best reported PR methodologies. Indeed, in some cases, its accuracy even exceeds the accuracy of the SVM.

Accuracy on Synthetic Data

We tested our approaches on three synthetic data sets described as follows and shown in Figure 5.2. Each data set has four classes and 100 two-dimensional points in each class. In the *SameVar* data, all classes have the same variance, while in *DiffVar*, the classes have different variances. *NonLinear* is a nonlinear data set.



(a) SameVar. (b) DiffVar. (c) NonLinear.

Figure 5.2: Plots of the synthetic data sets. This is a color figure, thus the readability may be affected if printed in grayscale.

Table 5.1: Summary of our NB methods and the benchmark methods used.

Category	**Method**	**Description**
Our Methods	ν-NB	ν is the lower bound of fraction of support vectors and upper bound of the fraction of error. Here, we invoke the decision rule specified by Equation (5.6).
	ν-NBN	Here, ν-NB uses the normalized distance as defined by Equation (5.7).
	α-NB	Here α is the fraction of support vectors, and we invoke the decision rule specified by Equation (5.6).
Generative	naive Bayes	This rule has only been used on artificial data. It may fail on real data.
Discriminative	NN	This is the nearest neighbor rule [155]. Here, we replace the inner product in the Euclidean distance with the RBF kernel, since the latter does not change the NN. Thus, we have invoked the kernelized NN rule.
	NC	This is the nearest centroid (or prototype) [156] rule. Again, we extended it to the kernelized version.
	NS	This is a Nearest Subspace method proposed in [109] (originally called the *linear regression classifier*). Since this method only works safely under the condition that the number of features must be greater than the class-sample-size, we again extended it into the kernelized version in order to let it operate under all conditions.
	SVM	In this case, we used the ν-SVM [157] (see Appendix B), where the one-versus-rest scheme and softmax function are used for the multi-class task.

For the artificial data, we compared our approaches with the naive Bayes [155], NN [155], NC [156], and SVM [157] classifiers. The linear kernel was used for the NBs, NC, and SVM on the first two data sets, and the *radial basis function* (RBF) kernel was used on the last data set. We ran a three-fold cross-validation on each data set 20 times. All the classifiers used the same training and testing splits in order to maintain a fair comparison. From the 20 results, we computed the mean and standard deviation (STD) of the accuracies. The results are illustrated in Figure 5.3. The full result can be found in Table 5.2.

Table 5.2: Results of the accuracies achieved by a three-fold cross-validation using the new and benchmark algorithms on the artificial data sets.

Data	ν-NB	ν-NBN	α-NB	Naive Bayes	NN	NC	SVM
SameVar	0.8716(0.0078)	0.8751(0.0048)	0.8753(0.0065)	0.8764(0.0038)	0.8121(0.0147)	0.8738(0.0033)	0.7765(0.0238)
DiffVar	0.9170(0.0056)	0.9107(0.0057)	0.9175(0.0060)	0.9314(0.0039)	0.8929(0.0086)	0.8959(0.0027)	0.8430(0.0270)
Nonlinear	0.9788(0.0049)	0.9749(0.0054)	0.9791(0.0048)	0.9264(0.0165)	0.9818(0.0037)	0.9408(0.0118)	0.9881(0.0031)

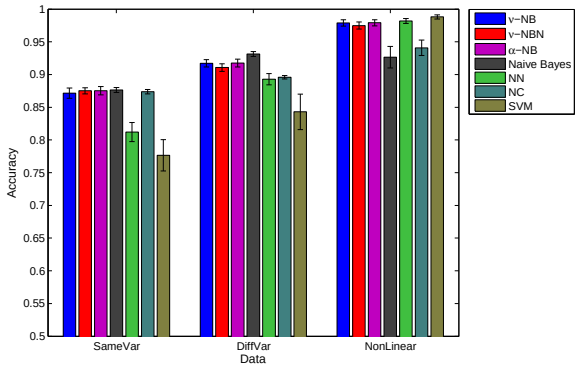


Figure 5.3: Performance on synthetic data. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

On the *SameVar* data, firstly, we can see that there is no significant difference between the ν-NB and ν-NBN, and α-NB. All of them yielded an almost-equivalent accuracy as the naive Bayes. Secondly, it can be seen from Figure 5.2a that the NB was able to identify the centers of each class accurately. The borders have the same volume, which demonstrates that the NB can identify the borders consistent with the variances. The NB approaches yielded an accuracy similar to the NC, which is reasonable because the identical variance of all classes is of no consequence to the NB. Thirdly, the NN and SVM do not obtain comparable results. This is because the distance measure of the NN is affected by noise, and the SVM is not able to "disentangle" each class well using a one-versus-all scheme.

On the *DiffVar* data, first, we see that the results again confirm that the NB can identify the borders consistent with the variances (see Figure 5.2b). The mean accuracies of all the NB approaches were very close to the naive Bayes classifier. However, the NC yielded worse results than the NB. This is because the variance information helped the NB, while the NC scheme did not consider it.

Finally, for the *NonLinear* data, firstly, we affirm that all our NB methods and the SVM yielded comparably good results. Secondly, the naive Bayes did not work well this time, because the data is not Gaussian. Further, the kernel NC was not competent either, because the data in the high-dimensional feature space have different variances for all the classes.

Performance on Real-Life Data

In order to demonstrate the performance of our NB approaches, we also compared them with benchmark approaches on 17 various data sets from bioinformatics, image processing, and so on. These data sets are summarized in Table 5.3.

Table 5.3: The real-life data sets used in our experiments.

Data	#Class	#Feature	#Sample
DNA [158]	3	180	2000
ExYaleB[159]	38	32256	2432
Ionosphere [160]	2	34	351
Iris [160]	3	4	150
Letter [158]	26	16	15000
MFEAT [160]	10	649	2000
Minsteries [161]	2	400	326
Pendigit [160]	10	16	10992
Pima [160]	2	8	768
Satimage [158]	6	36	4435
Segment [158]	7	19	2310
Svmguide2 [162]	3	20	391
Svmguide4 [162]	3	20	391
USPS [163]	10	256	9299
Vehicle [158]	4	18	846
Vowel [160]	11	10	990
Wave2 [160]	3	40	5000

Methods and Parameters: In this set of experiments, we included the ν-NB and the ν-NBN in the competition. However, we did not involve the α-NB on the real-life data sets, because it would have yielded the same performance as the ν-NB when the parameter (ν in ν-NB or α in the α-NB) is selected by inner three-fold cross-validation on the training set. The benchmark methods included the NN, NC, NS, and the SVM. In this set of tests involving real-life data, we did not include the naive Bayes classifier because it failed on some of them. Again, we used the RBF kernel in our schemes and in all the benchmark classifiers. All the parameters in each method were selected by a grid or a line-search based on the inner three-fold cross-validation accuracy of the training set. For ν-NB and ν-NBN, the range of ν was tested from the range $\max(0.025, \frac{1}{\frac{2}{3}s})$ to 0.95 by using a step-size of 0.025, where s was the mean class-sample-size of the training set. For the ν-SVM, the range of ν was from $\max(0.025, \frac{1}{\frac{2}{3}s})$ to $\min(f, 0.95)$, where f was the maximum feasible value of ν defined in [164]. For all the other methods except the NN, the parameter σ was searched for from 2^{d-2} to 2^{d+2} by involving a step-size 0.5 in the power, where $d = \log_2(\sqrt{m})$ (where

m is the number of features). This was inspired by LIBSVM [165] which sets the default value of σ to be 2^d.

The results of the accuracies of achieving a three-fold cross-validation using the new and benchmark algorithms on the real-life data sets are given in Table 5.4 and plotted in Figure 5.4. The results that we achieve in this case, seem to categorically demonstrate the power of the scheme. All the two NB algorithms are almost always better than all the other benchmark algorithms, except the SVM. This is not too difficult to understand because the SVM utilizes the information gleaned by invoking the borders from both the classes. As opposed to this, the NB border merely concentrates on the border that the testing sample is nearest to. The crucial issue that these results communicate is the fact that the NB strategy that we have proposed is a viable and competitive solution, and lends credibility to the fact that the new concept that one can use "borders" (or outliers) to achieve very accurate and almost-optimal PR.

Table 5.4: Results of the accuracies achieved by a three-fold cross-validation using the new and benchmark algorithms on the real-life data sets.

Data	ν-NB	ν-NBN	NN	NC	NS	SVM
DNA	0.7955	0.7635	0.6990	0.8915	0.4525	0.9385
EYaleB	0.7430	0.7364	0.7455	0.0259	0.0263	0.9239
Ionosphere	0.8632	0.8746	0.8604	0.7920	0.8063	0.9402
Iris	0.9267	0.9067	0.9333	0.8733	0.7000	0.9467
Letter	0.9248	0.9245	0.9352	0.7209	0.0517	0.9157
MFEAT	0.9640	0.9640	0.9800	0.9455	0.5200	0.9745
Minsteries	0.6258	0.6595	0.6043	0.4509	0.6472	0.7454
Pendigits	0.9829	0.9823	0.9929	0.8686	0.9925	0.9944
Pima	0.7227	0.7240	0.6810	0.7344	0.7005	0.7578
Satimage	0.8638	0.8634	0.8970	0.7932	0.9042	0.8992
Segment	0.9065	0.9065	0.9558	0.8476	0.2069	0.9468
Svmguide2	0.7877	0.7852	0.7161	0.7903	0.7212	0.8031
Svmguide4	0.6601	0.6405	0.6618	0.5376	0.3644	0.7598
USPS	0.8635	0.8621	0.9525	0.1167	0.6698	0.9505
Vehicle	0.7139	0.7128	0.6950	0.5816	0.2388	0.8002
Vowel	0.9434	0.9434	0.9646	0.8182	0.9697	0.9455
Wave2	0.8492	0.8484	0.7222	0.8086	0.6712	0.8538

Interpretation of the Results: With regard to the interpretation of the results, we state:

- First of all, as can be seen from the results, the difference between the ν-NB and the ν-NBN is negligible. However, ν-NB has a marginally higher rank than the ν-

NBN. Therefore, we can state that using an enhanced distance measure, as defined in Equation (5.7), is beneficial.

- Second, the SVM obtained the highest rank. However, by using Friedman test [85], there is no significant difference among the SVM, the NN, and the ν-NB under the current significant level. This is quite a remarkable conclusion.
- Third, the performances of NC and NS are very close to each other.
- Last, if we examine the accuracies of the classifiers, we can clearly identify two distinct groups: {SVM, NN, ν-NB, ν-NBN}, and {NC, NS}, demonstrating that our newly-introduced NB schemes are competitive to the best reported algorithms in the literature.

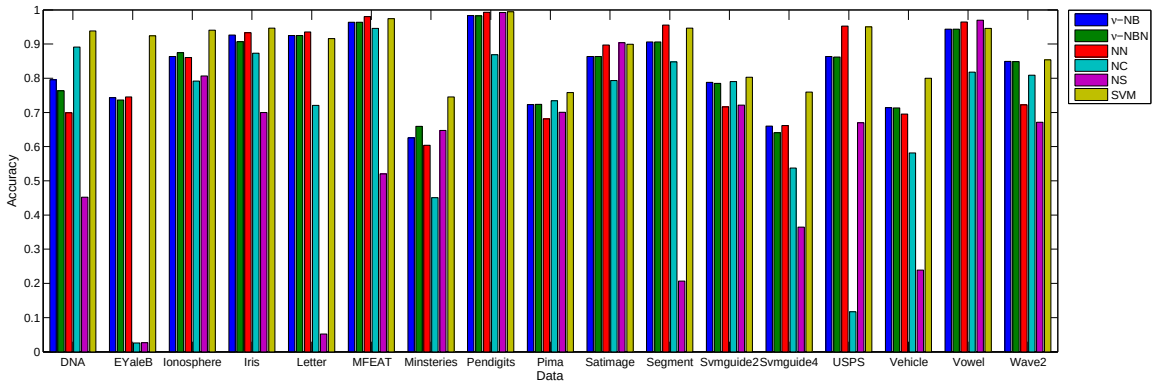


Figure 5.4: The accuracies achieved on three-fold cross-Validation for the 17 real-life data sets. This is a color figure, thus the readability may be affected if printed in grayscale. The order of the bars, from left to right, in the figure, is the same as these from top to bottom in the legend.

We also tested the performance of our NB method on the microarray data proposed in [33]. However, its performance was not good enough. We think it is because of the following reasons. First, the NB method learns the border of each class separately, that is the distributions of other classes are unseen. Thus, it needs a large number samples to learn the domain of a class. However, usually there is a small number of samples in each class of a microarray data set. Second, the distribution of a class of a microarray data set may be complex. For example, the class of breast tumor is in fact composed of many subtypes which have their own genotypical features. There are two possible solutions to overcome

these issues. First, supervised SVDD can be devised so that the data of the centers of others classes can serve as references. Second, proper clustering can be conducted for some classes that may be complexly distributed. This idea is similar to our local method described in Chapter 2.

We have introduced a new paradigm for pattern recognition which has not been reported in the literature earlier, which we shall refer to as the nearest border paradigm. This paradigm can be contrasted with the reported and existing PR paradigms such as the optimal Bayesian, kernel-based methods, nearest neighbor methods, nearest centroid methods, among others. The philosophy for developing such a NB strategy is also quite distinct from what has been used in the existing literature, because we shall attempt to create borders for each individual class *only* from the training data sets of *that class*. Indeed, unlike the traditional border identification methods, we have not achieved this by using *inter*-class criteria, but by searching for the border for a specific class in the d-dimensional hyper-space by invoking *only* the properties of the samples *within that class*. This has been, in turn, achieved, using the corresponding one-class SVM-based classifers. Once these borders have been obtained, we advocate that testing is accomplished by assigning the test sample to the class whose border it lies closest to. We emphasize that our methodology is actually counter-intuitive, because unlike the centroid or the median, these border samples are often "outliers" and are, indeed, the points that represent the class the least.

3 A hierarchical Model [3]

We have proposed our nearest border method above for multi-class classification. We also mentioned that a sufficient number of samples are required by each class for a satisfactory performance of the nearest border method. In this section, we propose another strategy – the hierarchical model, which extends linear models for multi-class classification problems. Our experiments on microarray data of breast tumor subtypes, show that our hierarchical model generalizes very well for a small number of samples.

3.1 Training Phase

We give an example of such a model to illustrate our method in Figure 5.5. Suppose there are five classes, namely $\{C_1, \cdots, C_5\}$. The training set is represented by a $m \times n$ matrix $\boldsymbol{D} = \{\boldsymbol{D}_1, \cdots, \boldsymbol{D}_5\}$ corresponding to the five classes. $\boldsymbol{D}_i$, of size $m \times n_i$, is the training data for class C_i. m is the number of features and n_i is the number of samples in class

[3] This section is based on our collaborative work published as [166].

C_i. $n = \sum_{i=1}^{5} n_i$ is the total number of training samples from all five classes. First of all, feature selection and classification are conducted, in a cross-validation fashion, for each class against the other classes. For example, suppose class C_3 obtains the highest rank based on its one-versus-rest accuracy. We thus record the list of the particular features selected and create a leaf for that class. We then remove the samples of the class, which results in $\boldsymbol{D} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{D}_4, \boldsymbol{D}_5\}$ and continue the process in the same fashion. Thus, at the second level, class C_5 yields the highest rank, and hence its feature list is retained and a leaf is created. Afterward the training data set becomes $\boldsymbol{D} = \{\boldsymbol{D}_1, \boldsymbol{D}_2, \boldsymbol{D}_4\}$ for the third level. We repeat the training procedure in the same fashion until there is no class to classify. At the last level, two leaves are created, for C_4 and C_2, respectively.

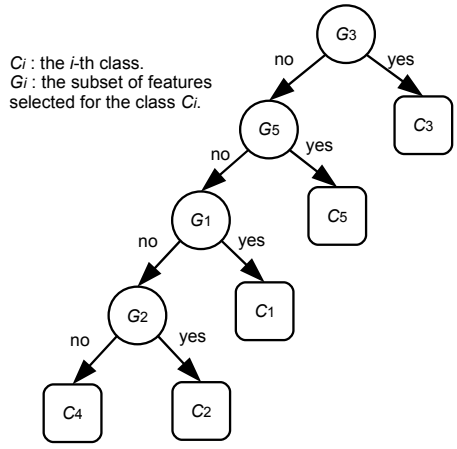


Figure 5.5: An example of the hierarchical model.

3.2 Prediction Phase

Once the training is complete, we can apply the scheme to predict the classes of new samples. Given the data of a sample, a sequence of classification steps are performed by tracing a path from the root of the tree toward a leaf. At each node in the path, only the features selected in the training phase are tested. The process starts at the first level (root of the tree), in which case only the features selected for C_3, namely G_3 are tested. If the new sample is classified as a positive sample, then the prediction outcome is class C_3, and the prediction phase terminates. Otherwise, the sequence of classification tests is performed in the same fashion, until a leaf is reached, in which case the prediction outcome is the class associated with the leaf that has been reached.

3.3 Characteristics of The Method

Our structured model has the following characteristics. First, it involves a greedy scheme that tries the subtype which obtains the most reliable prediction and the smallest number of genes first. Second, it can conduct feature selection and classification simultaneously. Essentially, it is a specific type of decision tree for classification. The differences between the proposed model and the traditional decision tree include: i) each leaf is unique, while one class usually has multiple leaves in the latter; ii) classifiers are learned at each node, while the traditional scheme learns decision rules; and iii) multiple features can be selected, while in the traditional scheme each node corresponds to only one feature. Third, the proposed model is flexible as any feature selection method and classifier can be embedded. Obviously, a classifier that can select features simultaneously also applies, (e.g. the l_1-norm SVM in Section 3.1 of Appendix B).

3.4 An Application to Predict Breast Cancer Subtypes

We have implemented the hierarchical model in MATLAB. Many classifiers and feature selection methods can be embedded in the hierarchical model, resulting in many instances of this model. We are interested in two instances of the hierarchical model: by using C-SVM and l_1-norm SVM, respectively (see Sections 2.3 and 3.1 of Appendix B for an introduction to C-SVM and l_1-norm SVM, respectively). For narrative convenience, we call them *hierarchical C-SVM* and *hierarchical* l_1*-norm SVM*, respectively. The former uses all features, while the latter conducts feature selection and classification simultaneously. We compared the former with one-versus-all C-SVM, and the latter with one-versus-all l_1-norm SVM. The parameters of the respective methods were selected by coarse grid or line search due to time constraint. Four-fold cross-validation was run 20 times for each method, and the average results were compared.

We tested them on the microarray gene expression data of five breast tumor subtypes, generated by Hu *et al.* [31]. Hu's data (GSE1992) were generated by three platforms including Agilent-011521 Human 1A Microarray G4110A (feature number version) (GPL885), Agilent-012097 Human 1A Microarray (V2) G4110B (feature number version) (GPL887), and Agilent Human 1A Oligo UNC custom Microarrays (GPL1390). Each platform has 22,575 probe sets, and there are 14,460 common probe sets among these three platforms. We used SOURCE [167] to obtain 13,582 genes with unique unigene IDs in order to merge data from different platforms together. It contains 158 samples from five subtypes (13 Normal, 39 Basal, 22 Her2, 53 LumA and, 31 LumB). The sixth subtype Claudin is excluded

from our current analysis as the number of samples of this class is too few (only five). However, we will investigate this subtype in our future work.

Table 5.5 shows the mean accuracies and standard deviations of our experiment. In this table, "OVA" and "HK" stand for "one-versus-all" and "hierarchical", respectively. The accuracy of a subtype is defined as the ratio of the number of correctly predicted test samples from this class to the total number of test samples from this class. The overall accuracy is defined as the ratio of the number of correctly predicted test samples to the total number of test samples. First of all, when using C-SVM, the hierarchical strategy obtained a better accuracy than the one-versus-all strategy. The difference between both accuracies is greater than their standard deviations. We can also observe an improvement in the accuracies of LumA and LumB by using the hierarchical model. Second, when using the l_1-norm SVM to select genes and classify samples simultaneously, the accuracy of the hierarchical scheme is also slightly higher than the one-versus-all scheme. Furthermore, the accuracies of LumA and LumB are also increased by using the hierarchical scheme, compared with the one-versus-all based l_1-norm SVM. However, the accuracy of Her2 obtained the hierarchical scheme is lower than that by one-versus-all scheme, which requires further investigation.

Table 5.5: The accuracy of the hierarchical model.

Method	Basal	Her2	LumA	LumB	Normal	Accuracy
OVA C-SVM	1.0000(0)	0.7773(0.0404)	0.7792(0.0392)	0.5435(0.0424)	0.7769(0.1112)	0.7870(0.0145)
HK C-SVM	0.9923(0.0118)	0.7750(0.0393)	0.8198(0.0436)	0.5952(0.0554)	0.7769(0.1138)	0.8085(0.0206)
OVA l_1-SVM	0.9987(0.0056)	0.8205(0.0528)	0.7868(0.0311)	0.5161(0.0603)	0.5962(0.1189)	0.7750(0.0154)
HK l_1-SVM	0.9962(0.0092)	0.7455(0.0526)	0.8132(0.0473)	0.5710(0.0662)	0.5846(0.1250)	0.7826(0.0300)

4 The Regularized Linear Models and Kernels Toolbox

We have implemented most of the methods mentioned in this chapter and Appendix B. Table 5.6 summarizes our current implementations, which constitute our *regularized linear models and kernels* (RLMK) toolbox. It can be downloaded at `https://sites.google.com/site/rlmktool`. We will continue realizing the rest and latest linear models, so that our RLMK toolbox can grow to a powerful and popular package.

Table 5.6: Current implementations of our RLMK toolbox.

Method	Optimization	Training Func.	Prediction Func.	Example
C-SVM	QP	CSVMQPTrain	SVMBinaryPredict	exampleSVMTwo
C-SVM	SMO	CSVMSMOTrain	SVMBinaryPredict	exampleSVMTwo
ν-SVM	QP	nuSVMQPTrain	SVMBinaryPredict	exampleSVMTwo
Two-class SVM	-	SVMBinaryTrain	SVMBinaryPredict	exampleSVMTwo
Multi-class SVM	-	SVMMultiTrain	SVMMultiPredict	exampleSVMMulti
Hard Margin l_2-Reg.	Closed-Form	khdlmTrain	khdlmPredict	exampleHDLM
l_1-norm SVM	LP	l1CSVMTrain	l1CSVMPredict	examplel1SVM
SVM-RFE	-	svmrfemrmr	-	exampleSVMRFE
Hypersphere SVDD	QP	SVDD	-	exampleSVDD
Hypersphere SVDD	SMO	SVDD	-	exampleSVDD
hierarchical Model	-	hierarchialModelTrain	hierarchialModelPredict	exampleHierarchialModel
Multi-class SVDD	-	SVDDTrain	SVDDPredict	exampleNBSVDD

5 Conclusions

In this chapter, we propose two strategies to extend sparse regularized linear models for multi-class classification problems. The first model is named nearest border model which learns the border of each class. The second model is a hierarchical model, which learns a specific tree structure. Our experiments show that the nearest border method performs well when there are a sufficient number of samples for each class in order to learn the border precisely, and the hierarchical model works very well with a small number of samples. Furthermore, we have implemented both methods and many regularized linear models in MATLAB. Our implementations can be download online. As a future work, we plan to improve the nearest border method by using supervised support vector domain description and local learning methods.

Publications

1. The hierarchical model for multi-class data was preliminary presented in [*I. Rezaeian, Y. Li, M. Crozier, E. Andrechek, A. Ngom, L. Rueda, and L. Porter, "Identifying informative genes for prediction of breast cancer subtypes," IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB), Nice, June, 2013, LNBI 7986, pp. 138-148.*].

2. The nearest border method for multi-class classification was proposed in [*Y. Li, B.J. Oommen, A. Ngom, and L. Rueda, "A new paradigm for pattern classification: nearest border techniques," 26th Australasian Joint Conference on Artificial Intelligence, New Zealand, Dec. 2013.*].

3. The RLMK toolbox is publicly available [`https://sites.google.com/site/rlmktool`], but has not been formally published yet.

Chapter 6

Spectral Method for Clustering Microarray Time-Series Data [1]

1 Introduction

A common problem in molecular biology and other research areas involves partitioning a set of experimental data into clusters in such a way that the data points within the same cluster are highly similar, while data points in different clusters are as dissimilar as possible. In particular, an important process in functional genomic studies is to cluster microarray time-series data, where genes with similar expression profiles are expected to be functionally related [2]. This problem has its own features that makes it different from general clustering problems, namely exchanging two time points can make the resulting clusters completely meaningless. In this direction, many approaches have been proposed for clustering microarray time series data. Brief descriptions of all these methods are discussed below.

A Bayesian approach in [169], a partitional clustering based on k-means in [170], and a Euclidean distance approach in [171] have been proposed for clustering gene expression time-series profiles. They have applied *self-organizing maps* (SOMs) to visualize and interpret the gene temporal expression profile patterns. A hidden phase model was used for clustering time-series data to define the parameters of a mixture of normal distributions in a Bayesian-like manner that are estimated by using *expectation maximization* (EM) [172]. Also, the methods proposed in [173, 174] are based on correlation measures. A method that uses jack-knife correlation with or without using seeded candidate profiles was proposed for

[1] This chapter is based on our publication [168].

clustering time-series microarray data as well [174], where the resulting clusters depend upon the initially chosen template genes, because there is a possibility of missing important genes. Another method proposed for these types of data is a regression-based method [175], which was proposed to address the challenges in clustering short time-series expression data.

Analyzing temporal gene expression profiles that are non-uniformly sampled and which may contain missing values has been studied in [176], while clustering temporal gene expression profiles was studied by identifying homogeneous clusters of genes in [177]. In that work, the *shapes of the curves* were considered instead of the *absolute expression ratios*. Fuzzy clustering of gene expression profiles has been studied in [178], where the similarities between co-expressed genes are computed based on the rate of change of the expression ratios across time. In [179], the idea of order-restricted inference levels across time was applied to select and cluster genes, where the estimation makes use of known inequalities among parameters. In that approach, two gene expression profiles fall into the same cluster, if they show similar profiles in terms of directions of the changes of expression ratios, regardless of how big or small the changes are. In [180], pairs of profiles represented by piecewise linear functions are aligned in such a way to minimize the integrated squared area between the profiles. An agglomerative clustering method combined with an area-based distance measure between two aligned profiles was used to cluster microarray time-series data. The pairwise gene expression profile alignment approach of [180] was re-formulated in [181] in terms of integrals of arbitrary functions, in particular, by using natural cubic spline interpolations. The pairwise alignment formulae of [180] from the case of piecewise linear profiles is generalized to profiles that are any continuous integrable functions on a finite interval. Afterwards, the concept of pairwise alignment was extended to multiple expression profile alignment, where profiles from a given set are aligned in such a way that the sum of integrated squared errors over a time-interval and defined on the set is minimized. Finally, combining k-means clustering with multiple alignment to cluster microarray time-series data, produced a high-quality clustering for 221 pre-clustered *Saccharomyces cerevisiae* gene expression time-series profiles, leading to an accuracy of 79.64%, when the data are classified by a k-nearest neighbor in a 10-fold cross-validation setup.

Along with the developments on alignment-based gene time-series analysis, other techniques have also been moving forward. In this direction, a new fuzzy cover based clustering approach was proposed in [182]. That approach, combined with a variation-based co-expression detection process, was later applied to clustering gene expression time-series data [183], which is known as the *variation-based coexpression detection* (VCD) algorithm. Both the alignment-based and the variation-based clustering algorithms have been reported

to obtain good performance on various data sets. The principle behind these two approaches is to first transform the data in such a way that the *shapes* of any two expression profiles can be compared in the subsequent clustering step. However, the performance of these two methods has been quantified in different ways and both performed different clustering algorithms on their respective transformed data. In [184], we compared the VCD and two alignment-based clustering methods, k-MCMA and EMMA, on two yeast data sets, and reported a superior performance of the alignment-based approaches.

Contributions: In this chapter, unlike in the comparison studies presented in [184], we study the abilities of the alignment-based and the variation-based transformations to yield high-quality clustering results. Both transformation methods are compared using the same sparse clustering technique, the *spectral clustering* algorithm, and on the same data sets. In [184], we were interested in comparing different clustering methods whereas here, we are interested in comparing the aforementioned two transformation methods. In order to assess the quality of the resulting clusters for a given transformation method, a new cluster validity index is also proposed in this chapter. Thus, the contributions of this chapter are three-fold:

1. We apply spectral clustering algorithms to expression time-series analysis.
2. We propose new measurements for the quality of the spectral clustering approach.
3. we empirically show that when applied to two well-known data sets, the alignment-based transformation yields better clustering results than the variation-based transformation.

To the best of our knowledge, the proposed approaches mentioned in 1 and 2 have not been used in the past for this specific problem.

2 Alignment-Based Data Transformation

Alignment-based transformation methods have been proposed in order to consider vertical shifting of profiles in such a way that shape similarities between two or more profiles can be detected. A more general case of this alignment takes into account the lengths of the intervals, which is accomplished by means of analyzing the area between two expression profiles, joined by the corresponding measurements at subsequent time points. This is equivalent to consider the sum or average of squared errors between the infinite points in

the two lines. This analysis can be easily achieved by computing the underlying integral, which is analytically resolved in advance, subsequently avoiding expensive computations during the clustering process.

Given two profiles, $x(t)$ and $y(t)$ (either piece-wise linear or continuously integrable functions), where $y(t)$ is to be aligned with $x(t)$, the basic idea of alignment is to *vertically shift* $y(t)$ towards $x(t)$ in such a way that the *integrated squared errors* between the two profiles is minimal. Let $\hat{y}(t)$ be the result of shifting $y(t)$. Here, the *error* is defined in terms of the area between $x(t)$ and $\hat{y}(t)$ in interval $[0, T]$. Functions $x(t)$ and $\hat{y}(t)$ may cross each other many times, but the sum of all the areas in which $x(t)$ is above $\hat{y}(t)$ minus the sum of those areas in which $\hat{y}(t)$ is above $x(t)$, should be minimal (see Figure 6.1). Let a denote the amount of vertical shifting of $y(t)$. Then, the value $a_{\min}$ of a, which minimizes the integrated squared error between $x(t)$ and $\hat{y}(t)$, should be found. Once $a_{\min}$ is obtained, the alignment process consists of performing the shift on $y(t)$ as $\hat{y}(t) = y(t) - a_{\min}$.

The authors of [181] generalized the pairwise alignment model of [180] from piece-wise linear profiles to profiles which are *any* integrable functions on a finite interval. Suppose that one has two such profiles, $x(t)$ and $y(t)$, defined on the time-interval $[0, T]$. The alignment process consists of finding the value of a that minimizes:

$$f_a(x, y) = \int_0^T [x(t) - [y(t) - a]]^2 \, dt. \tag{6.1}$$

Setting the derivative $\frac{d}{da} f_a(x(t), y(t)) = 0$, and solving for a, yields the solution:

$$a_{\min} = -\frac{1}{T} \int_0^T [x(t) - y(t)] dt, \tag{6.2}$$

and since $\frac{d^2}{da^2} f_a(x, y) = 2T > 0$ then $a_{\min}$ is a minimum. The integrated error between $x(t)$ and the shifted $\hat{y}(t) = y(t) - a_{\min}$ is then

$$\int_0^T [x(t) - \hat{y}(t)] dt = \int_0^T [x(t) - y(t)] dt + a_{\min} T = 0. \tag{6.3}$$

In terms of profiles as shown in Figure 6.1, this means that the sum of all the areas where $x(t)$ is above $y(t)$ minus the sum of those areas where $y(t)$ is above $x(t)$ is zero.

Given an original profile $x(t) = [e_1, e_2, \ldots, e_n]$ (with n expression values taken at n time-points $t_1, t_2, \ldots, t_n$), the *natural cubic spline* interpolation is used, with n knots,

$(t_1, e_1), \ldots, (t_n, e_n)$, to represent $x(t)$ as a continuously integrable function as follows:

$$x(t) = \begin{cases} x_1(t) & \text{if} \quad t_1 \leq t \leq t_2 \\ & \vdots \\ x_{n-1}(t) & \text{if} \quad t_{n-1} \leq t \leq t_n \end{cases} \tag{6.4}$$

where $x_j(t) = x_{j3}(t - t_j)^3 + x_{j2}(t - t_j)^2 + x_{j1}(t - t_j)^1 + x_{j0}(t - t_j)^0$ interpolates $x(t)$ in interval $[t_j, t_{j+1}]$, with spline coefficients $x_{jk} \in \Re$, for $1 \leq j \leq n-1$ and $0 \leq k \leq 3$.

For practical purposes, given the coefficients $x_{jk} \in \Re$, associated with $x(t) = [e_1, e_2, \ldots, e_n] \in \Re^n$, one needs to transform $x(t)$ into a new space as $x(t) = [x_{13}, x_{12}, x_{11}, x_{10}, \ldots, x_{j3}, x_{j2}, x_{j1}, x_{j0}, \ldots, x_{(n-1)3}, x_{(n-1)2}, x_{(n-1)1}, x_{(n-1)0}] \in \Re^{4(n-1)}$. One can add or subtract polynomials given their coefficients, and the polynomials are continuously differentiable. This yields the following analytical solution for $a_{\min}$:

$$\begin{aligned} a_{\min} &= -\frac{1}{T} \sum_{j=1}^{n-1} \int_{t_j}^{t_{j+1}} [x_j(t) - y_j(t)] dt \\ &= -\frac{1}{T} \sum_{j=1}^{n-1} \sum_{k=0}^{3} \frac{(x_{jk} - y_{jk}) (t_{j+1} - t_j)^{k+1}}{k+1}. \end{aligned} \tag{6.5}$$

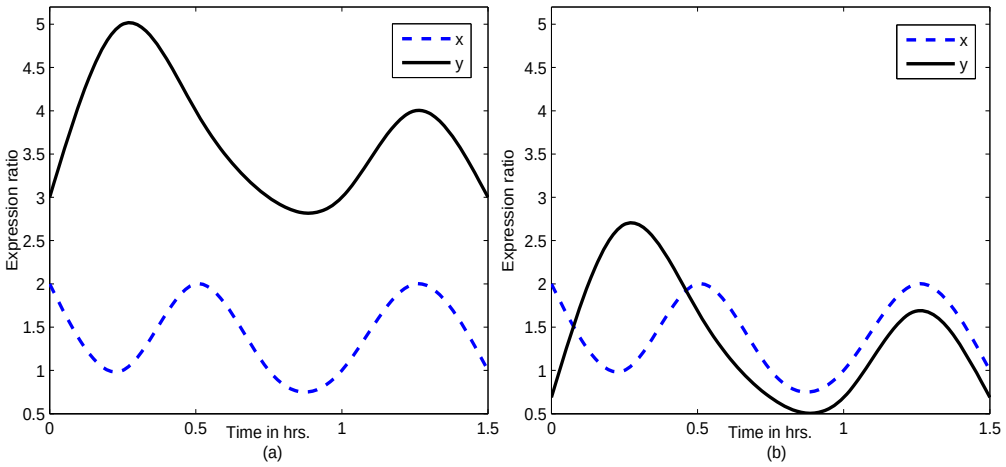


Figure 6.1: (a) Unaligned profiles $x(t)$ and $y(t)$. (b) Aligned profiles $x(t)$ and $y(t)$, after applying $y(t) \leftarrow y(t) - a_{\min}$.

Figure 6.1(b) shows a pairwise alignment of the two profiles of Figure 6.1(a), after applying the vertical shifting $y(t) \leftarrow y(t) - a_{\min}$. It can be noticed that the two aligned profiles cross each other many times. As a consequence of this, the integrated error as per

Equation (6.3) is zero. In particular, from Equation (6.3), the horizontal t-axis will bisect a profile $x(t)$ into two halves with equal areas when $x(t)$ is aligned to the t-axis. This property is quite important to define the multiple alignment of a set of profiles, as shown below.

Given a data set $X = \{x_1(t), \ldots, x_s(t)\}$, the profiles should be aligned in such way that the integrated squared error between any two *vertically shifted* profiles is minimal. Thus, for any $x_i(t)$ and $x_j(t)$, the values of a_i and a_j can be found via minimizing:

$$\begin{aligned} f_{a_i,a_j}(x_i, x_j) &= \int_0^T [\hat{x}_i(t) - \hat{x}_j(t)]^2 dt \\ &= \int_0^T [[x_i(t) - a_i] - [x_j(t) - a_j]]^2 dt\,, \end{aligned} \tag{6.6}$$

where *both* $x_i(t)$ and $x_j(t)$ are shifted vertically by a_i and a_j, respectively, in possibly different directions, whereas in the pairwise alignment of Equation (6.1), profile $y(t)$ is shifted towards the *fixed* profile $x(t)$. The multiple alignment process consists then of finding the values of $a_1, \ldots, a_s$ that minimize:

$$F_{a_1,\ldots,a_s}(x_1, \ldots, x_s) = \sum_{1 \le i < j \le s} f_{a_i,a_j}(x_i, x_j)\ . \tag{6.7}$$

The solution for each a_i of the above minimization problem is given by the following theorem derived in [181]:

Theorem 1 (Universal Alignment Theorem)**.** *Given a fixed profile, $z(t)$, and a set of profiles, $X = \{x_1(t), \ldots, x_s(t)\}$, there always exists a multiple alignment, $\hat{X} = \{\hat{x}_1(t), \ldots, \hat{x}_s(t)\}$, such that:*

$$\hat{x}_i(t) = x_i(t) - a_{\min_i},\ a_{\min_i} = -\frac{1}{T}\int_0^T [z(t) - x_i(t)]dt, \tag{6.8}$$

and, in particular, for profile $z(t) = 0$, defined by the horizontal t-axis, one has:

$$\hat{x}_i(t) = x_i(t) - a_{\min_i},\ \textit{where, } a_{\min_i} = \frac{1}{T}\int_0^T x_i(t)dt. \tag{6.9}$$

This theorem is quite important since it allows us to apply multiple alignment as a preprocessing step, and then an iterative clustering algorithm, such as k-means, as a second step. This implies a substantial improvement on efficiency in computations and provides independence of the clustering algorithm. The proof of this theorem and other related results can be found in [181].

We use the multiple alignment of Equation (6.9) in all subsequent discussions. Using spline interpolations, each profile $x_i(t)$, $1 \leq i \leq s$, is a continuous integrable profile:

$$x_i(t) = \begin{cases} x_{i,1}(t) & \text{if} \quad t_1 \leq t \leq t_2 \\ & \vdots \\ x_{i,n-1}(t) & \text{if} \quad t_{n-1} \leq t \leq t_n \end{cases} \tag{6.10}$$

where, $x_{i,j}(t) = x_{ij3}(t-t_j)^3 + x_{ij2}(t-t_j)^2 + x_{ij1}(t-t_j)^1 + x_{ij0}(t-t_j)^0$ represents $x_i(t)$ in interval $[t_j, t_{j+1}]$, with spline coefficients x_{ijk} for $1 \leq i \leq s$, $1 \leq j \leq n-1$ and $0 \leq k \leq 3$. Thus, the analytical solution for $a_{\min_i}$ in Equation (6.9) is:

$$a_{\min_i} = \frac{1}{T} \sum_{j=1}^{n-1} \sum_{k=0}^{3} \frac{x_{ijk} \left(t_{j+1} - t_j\right)^{k+1}}{k+1} \tag{6.11}$$

Given a set of profiles $X = \{x_1(t), \ldots, x_s(t)\}$, a representative *centroid profile* $\mu(t)$, which well represents X, should be found. An obvious choice is the function that minimizes:

$$\Delta[\mu] = \sum_{i=1}^{s} d\left(x_i, \mu\right). \tag{6.12}$$

where, Δ plays the role of the *within-cluster-scatter* defined in [180], and the (L^2) distance between two profiles, $x(t), y(t)$, was defined in [181] as:

$$d(x,y) = \frac{1}{T} \int_0^T \left[\hat{x}(t) - \hat{y}(t)\right]^2 dt. \tag{6.13}$$

The distance $d(\cdot,\cdot)$ as defined in Equation (6.13) is unchanged by an additive shift $x(t) \rightarrow x(t) - a$ in either of its arguments, and hence, it is order-preserving; that is, $d(u,v) \leq d(x,y)$ if and only if $d\left(\hat{u},\hat{v}\right) \leq d\left(\hat{x},\hat{y}\right)$ [181]. Therefore, one has:

$$\Delta[\mu] = \sum_{i=1}^{s} d\left(\hat{x}_i, \mu\right) = \frac{1}{T} \int_0^T \sum_{i=1}^{s} \left[\hat{x}_i(t) - \mu(t)\right]^2 dt, \tag{6.14}$$

where, $\hat{X} = \{\hat{x}_1(t), \ldots, \hat{x}_s(t)\}$ is the multiple alignment of Equation (6.9). $\Delta[\mu]$ is a *functional* of μ; that is, a mapping from the set of real valued functions defined on $[0,T]$ to the set of real numbers. The solution for μ in Equation (6.14) was obtained in [181] by setting

the functional derivative[2] $\frac{\delta\Delta[\mu]}{\delta\mu(t)} = 0$ and then solve for μ as follows:

$$\mu(t) = \frac{1}{s}\sum_{i=1}^{s}\hat{x}_i(t). \tag{6.15}$$

With the spline coefficients, x_{ijk}, of each $x_i(t)$ interpolated as in Equation (6.10), the analytical solution for $\mu(t)$ in Equation (6.15) is:

$$\mu_j(t) = \frac{1}{s}\sum_{i=1}^{s}\left[\sum_{k=0}^{3} x_{ijk}\,(t-t_j)^k\right] - a_{\min_i}, \tag{6.16}$$

in each interval $[t_j, t_{j+1}]$. Thus, the centroid is the average point in the multidimensional space; that is, its coordinates are the arithmetic mean for each dimension separately over all the points in the cluster. In a sense, it is the center of gravity for the respective cluster. The distance between two clusters is determined as the difference between the two centroids. Equation (6.15) applies to aligned profiles, while Equation (6.16) can apply to unaligned profiles.

In [181, 186], an initial data set $\mathcal{X} = \{x_1(t), \ldots, x_s(t)\}$ is first transformed into a set of multiple-aligned profiles $\hat{\mathcal{X}} = \{\hat{x}_1(t), \ldots, \hat{x}_s(t)\}$, and then a given clustering method is applied. The k-means clustering was applied in [181] and expectation-maximization (EM) clustering was applied in [186]. These two clustering approaches are called *k-MCMA* and *EMMA*, respectively. It was also shown in [181] that: (i) any distance-based clustering method can be used; (ii) clustering the original data set $\mathcal{X}$ is equivalent to clustering the transformed data set $\hat{\mathcal{X}}$, provided that the centroids are initialized in a similar way; (iii) clustering multiple-aligned profiles is much faster than clustering the original data set.

3 Variation-Based Data Transformation

The authors of [183] proposed a novel clustering scheme called the *variation-based coexpression detection* (VCD) algorithm in order to analyze trends of expression profiles based on their variations between adjacent time points. The VCD, which does not require that the number of clusters be known in advance, also includes a criterion for calculating the degree of change of the expression between adjacent time points and evaluating the trend similarities between two profiles. As in the alignment-based clustering approaches, the VCD also

[2]For a functional $F[\phi]$, the functional derivative is defined as $\frac{\delta F[\phi]}{\delta\phi(t)} = \lim_{\epsilon\to 0}\frac{(F[\phi+\epsilon\delta_t]-F[\phi])}{\epsilon}$, where $\delta_t(\tau) = \delta(\tau - t)$ is the Dirac delta function centered at t. [185]

performs a transformation on the initial data set $\mathcal{X} = \{x_1(t), \ldots, x_s(t)\}$, and then proceeds to cluster the transformed data $\vec{\mathcal{X}} = \{\vec{x}_1(t), \ldots, \vec{x}_s(t)\}$. Each profile $x_i(t) = (x_{i1}, \ldots, x_{iT})$ is transformed into a variation vector $\vec{x}_i(t) = (\vec{x}_{i1}, \ldots, \vec{x}_{i(T-1)})$ of length $T-1$, where

$$\vec{x}_{ij} = x_{i(j+1)} - x_{ij} \tag{6.17}$$

for $j = 1, \ldots, T-1$. The amplitude of the expression change at a time point can be emphasized by making the following change to Equation (6.17)

$$\vec{x}_{ij} = \frac{t_{i2} - t_{i1}}{t_{i(j+1)} - t_{ij}} \times \frac{|x_{i(j+1)} - x_{i1}|}{x_{i1}} \times (x_{i(j+1)} - x_{ij}), \tag{6.18}$$

where t_{ij} is the time point corresponding to the gene expression ratio x_{ij}, and the first term at the right of the equation is to tackle the problem of unequal time intervals. Clearly, expression profiles that have similar trends or shapes will also have similar variation vectors (the opposite is not necessarily true), and hence subsets of co-expressed profiles can be obtained by clustering their variation vectors. In this regard, in [183] the cosine similarity function was used as follows:

$$\cos\langle \vec{x}_i(t), \vec{x}_j(t) \rangle = \frac{\vec{x}_i(t) \cdot \vec{x}_j(t)}{||\vec{x}_i(t)|| \times ||\vec{x}_j(t)||} \tag{6.19}$$

to measure the similarity between two variation vectors in their clustering process, where $||\cdot||$ is the Euclidean norm.

The subsequent clustering step of the VCD of [183] is based on the fuzzy cover clustering approach of [182], which aims to find a minimal set of covers that are fuzzy hyper-spheres centered each at a variation vector called *cover prototype*. The covers have the same radii, and a variation vector belongs to a cover if its cosine similarity with the cover prototype is greater than a predefined threshold λ. The prototypes are selected in such a way that the overlapping areas between their covers are minimized and the number of covers required to enclose the data set $\vec{\mathcal{X}}$ is also minimized; this is equivalent to the classical *vertex cover problem*, an NP-hard problem, for which in [183], the authors proposed a greedy algorithm to search for a minimal set of covers that encloses $\vec{\mathcal{X}}$. A variant of the agglomerative hierarchical clustering algorithm is then applied to the collection of covers: two covers are merged into a cluster, if they satisfy a merging criterion that takes into account the distance between these cover prototypes and the within-cover variance for each of these covers.

4 Time-Series Spectral Clustering

We applied a normalized spectral clustering algorithm to the transformed data sets, $\hat{\mathcal{X}}$ and $\vec{\mathcal{X}}$, obtained by performing the transformations of Equation (6.9) and Equation (6.17 or 6.18), respectively. An excellent tutorial about the family of spectral algorithms can be found in [187] for interested readers. Here, we briefly discuss the components of our approach, shown in Algorithm 6.1, for clustering gene time-series and refer readers to [187] for more details. Spectral clustering is an emerging clustering method which has many fundamental advantages over traditional methods. In many cases, it outperforms traditional clustering algorithms, it is simple to implement, it can be solved efficiently, and it does not make any assumptions on the shapes or forms of the clusters. Roughly speaking, spectral clustering finds a partition of a similarity graph defined on the input data set in such a way that the edges between different groups have very low weights (and hence, points in different clusters are different from each other), while the edges within a group have relatively high weights (and hence, points within the same clusters are similar to each other). The weighted adjacency matrix W is usually sparse. If it is sparse, then the graph Laplacian matrix L is sparse. It has been reported that if L is sparse, then its eigen-decomposition is very efficient [187]. Therefore, spectral clustering is a very fast sparse method for clustering large-scale data. This is our *first rationale* that we apply it to cluster microarray time-series data where there are usually thousands, even tens of thousands, of genes. Our *second rationale* is that the spectral clustering method is a kernel method as the input is a similarity matrix, rather than the original data. Thus, it is very flexible to cluster gene expression time series by using different (dis)similarity measures, for example the alignment-based and variation-based measures mentioned above.

The algorithm for spectral clustering microarray time-series data is depicted in Algorithm 6.1. The input $\bar{\mathcal{X}} = \{\bar{x}_1, \ldots, \bar{x}_n\}$ is either the transformation $\hat{\mathcal{X}}$ of Equation (6.9) or the transformation $\vec{\mathcal{X}}$ of Equation (6.17 or 6.18); S is the $n \times n$ similarity matrix containing the similarities between all pairs of points $\bar{x} \in \bar{\mathcal{X}}$; G is a weighted undirected graph with vertices $\{\bar{x}_1, \ldots, \bar{x}_n\}$, where $\bar{x}_i$ connected to $\bar{x}_j$ if $\bar{x}_i$ is among the c-nearest neighbors of $\bar{x}_j$, or if $\bar{x}_j$ is among the c-nearest neighbors of $\bar{x}_i$, and the edge carries a non-negative weight $w_{ij} = s(\bar{x}_i, \bar{x}_j) \geq 0$ which is the similarity between $\bar{x}_i$ and $\bar{x}_j$; $W = [w_{ij}]$ is the adjacency matrix; D is the diagonal *degree matrix* with the degrees $d_i = \sum_{j=1}^{n} w_{ij}$ of $\bar{x}_{ij}$, $1 \leq i \leq n$, in the main diagonal. It can be seen that each node connects to at least c nodes. Although any clustering method can be used in Step 7 of Algorithm 6.1 once the rows of V are computed, we used k-means due to its simplicity and fairly good efficiency. We fixed $\sigma = 1$ in our application. There is no systematic theoretical study on the choice of the parameter

Algorithm 6.1 *Time-Series Spectral Clustering*

Input: Transformed profiles, $\bar{\mathcal{X}} = \{\bar{x}_1, \ldots, \bar{x}_n\}$; Desired number of clusters, $k > 0$; Number of nearest neighbors, $c \geq 1$

Output: Clusters $\bar{\mathcal{C}}_1, \ldots, \bar{\mathcal{C}}_k$

1. $S \in \Re^{n \times n} \leftarrow$ similarity matrix of $\bar{\mathcal{X}}$;
2. $G \leftarrow c$-nearest neighbor similarity graph of S;
3. $W \leftarrow$ weighted adjacency matrix of G;
4. $L \leftarrow D - W$; *the un-normalized graph Laplacian*;
5. $V = (v_1, \ldots, v_k) \in \Re^{n \times k} \leftarrow$ the first k generalized eigenvectors of L; *see reference [187]*;
6. For $1 \leq i \leq n$, do: $r_i \in \Re^k \leftarrow i$-th row of V;
7. Cluster $\{r_1, \ldots, r_n\}$ into k clusters $\mathcal{R}_1, \ldots, \mathcal{R}_k$;

return Clusters $\bar{\mathcal{C}}_1, \ldots, \bar{\mathcal{C}}_k$ with $\bar{\mathcal{C}}_i = \{j \mid r_j \in \mathcal{R}_i\}$;

c. When choosing c in c-nearest neighbor, one should make sure that the similarity graph is connected, or only consists of "few" connected components and very few or no isolated vertices. A rule of thumb is choosing c on the order of $\log(n)$, where n is the number of samples. However, the choice of c depends upon data at hand. We found that $c = 10$ is a good choice. It is an open problem to choose K in clustering. There are many indices for general purpose [188]. One particular tool for spectral clustering is eigengap heuristic [187]. We invented the S_c index to test the quality of spectral clustering (see next section).

In terms of notation, our time-series spectral clustering algorithm is called SCMA if the alignment-based transformation is used, or SCVV if the variation-based transformation is used. The similarity measure used for matrix S is defined as:

$$s_{\text{MA}}(\bar{x}_i, \bar{x}_j) = e^{-\frac{d(\bar{x}_i, \bar{x}_j)}{2\sigma^2}}, \tag{6.20}$$

between two aligned profiles $\bar{x}_i$, $\bar{x}_j$ in the SCMA method, and $d(\bar{x}_i, \bar{x}_j)$ is the distance function defined in Equation (6.13). For the SCVV method, the similarity between the variation vectors is given by:

$$s_{\text{VV}}(\bar{x}_i, \bar{x}_j) = \frac{1}{2}(1 + \cos\langle \bar{x}_i, \bar{x}_j \rangle). \tag{6.21}$$

5 Cluster Validity

Two important issues when applying a clustering approach are (i) if the number of clusters is adequate for a specific data set, and (ii) how good is the clustering for a specific number of clusters in terms of *well-separated* and *compact* clusters. To determine the appropriate number of clusters and also the goodness or validity of the resulting clusters, our transformation-based spectral clustering approach is used in conjunction with a new cluster validity index, which we call the S_c index. Since the clusters resulting from spectral clustering may have arbitrary shapes (not necessarily hyper-spherical or hyper-ellipsoidal), we devised this index specially for assessing the results of that algorithm. Once the appropriate number of clusters is determined for a given transformation method, our transformation-based spectral clustering is used for proper partitioning of the data into the said number of clusters. Let K be the number of clusters. The cluster validity indices of [189] use within-cluster variances, between-cluster variances, or cluster diameter to assess the quality of a clustering result. As mentioned above, they work well for conventional clustering algorithms, such as k-means and EM, which in general produce clusters with hyper-spherical or hyper-ellipsoidal shapes in the Euclidean space. However, spectral clustering can also find clusters with irregular shapes. Therefore, the current indices are not suitable for evaluating the clustering produced by spectral methods. We thus propose here a new validity index, S_c, which is used for assessing the results of a spectral clustering method. For each spectral cluster, we create a complete weighted undirected graph where the nodes are the members of the cluster. Any two members are connected by an edge labeled with the distance between them. In our index, the compactness of a cluster is measured by computing the average weight of the minimum spanning tree of the cluster's graph. The separation between the clusters is measured by computing the average distance for all pairs of clusters. That is, given two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ with m members $\{\bar{c}_{i1}, \ldots, \bar{c}_{im}\}$ and n members $\{\bar{c}_{j1}, \ldots, \bar{c}_{jn}\}$, respectively, the distance between $\mathcal{C}_i$ and $\mathcal{C}_j$ is defined as follows:

$$d_C(\mathcal{C}_i, \mathcal{C}_j) = \frac{\sum_{k=1}^{m} d(\bar{c}_{ik}, \bar{c}_{js}) + \sum_{k=1}^{n} d(\bar{c}_{it}, \bar{c}_{jk})}{m+n}, \tag{6.22}$$

where $\bar{c}_{js}$ is the member of $\mathcal{C}_j$ closest to the member $\bar{c}_{ik}$ of $\mathcal{C}_i$, $\bar{c}_{it}$ is the member of $\mathcal{C}_i$ closest to the member $\bar{c}_{jk}$ of $\mathcal{C}_j$.

Our S_c index is defined as the ratio of compactness and separation as follows:

$$S_c(K) = \frac{\sum_{k=1}^{K} w_k / K}{\sum_{i=1}^{K-1} \sum_{j=i+1}^{K} d_C(\mathcal{C}_i, \mathcal{C}_j) / [K(K-1)]}, \tag{6.23}$$

where K is the number of clusters, and w_k is the average weight of the minimum spanning tree of the k-th cluster. The best number of clusters corresponds to the value of K for which the S_c is minimal.

6 Experimental Results and Analysis

In this section, the results of alignment-based and variation-based transformation methods are compared visually and quantitatively using spectral clustering, on two well-known data sets. The first data set, *Saccharomyces cerevisiae* [2], contains mRNA transcript ratios during the yeast cell cycle, and was used for analysis in [181] and [186]. The second data set includes progressions of the cell cycle fission yeast *Schizosaccharomyces pombe* [190], which was used for analysis in [184, 190].

The data set of pre-clustered budding yeast [2][3] contains gene expression time-series profiles of the complete characterization of mRNA transcript levels during the yeast cell cycle. These experiments measured the expression ratios of the 6,220 yeast genes during the cell cycle at seventeen different points, from 0 to 160 minutes, at every 10-minute time-interval. In [2], 221 of these profiles were analyzed separately, and are the ones that are analyzed in this chapter too. Each expression profile is normalized as in [2]; that is, each transcript ratio is divided by the mean value of each profile with respect to each other.

The data set contains five *known* clusters called *phases*: Early G1 phase (32 genes), Late G1 phase (84 genes), S phase (46 genes), G2 phase (28 genes) and M phase (31 genes); the phases are visualized in Figure 6.2(b). Setting $k = 5$, we applied our transformation-based spectral clustering to the data set to see if they are able to find these phases as accurately as possible. Next, we compare the resulting clusters with the five phases analyzed in [2].

The five yeast clusters found by the alignment-based (SCMA) and the variation-based (SCVV) spectral clustering methods are shown in Figure 6.2, respectively, as well as the five yeast phases. To measure the performance of each method, we assigned each of the five clusters of that method to a yeast phase using the *Hungarian algorithm* [191]. The Hungarian method is a combinatorial optimization algorithm that solves the assignment problem in polynomial time. Our phase assignment problem and the complete discussion of the solution can be found in [181]. In Figure 6.2, for each method: the cluster and the phase of each of the five selected pairs, found by the Hungarian algorithm, are shown at the same level; e.g., cluster *C1* of SCMA corresponds to the *Early G1* phase of [2] by our phase assignment approach, and hence they are at the same level in the figure. The horizontal

[3] http://genomics.stanford.edu/yeast_cell_cycle/cellcycle.html

axis represents the time-points in minutes and the vertical axis represents the expression values. Each cluster is vertically shifted up by three units to visually distinguish it from the others. The dashed black lines are the *learned cluster centroids* from a given method or the *known phase centroids* of the yeast data. In the figure, each cluster and phase were multiple-aligned to enhance visualization.

Figure 6.2 clearly shows a high degree of similarity between the clusters found by the two algorithms and the yeast phases. Visually, each cluster for each method is *very similar* to one of the yeast phases (i.e., the one shown at the same level). Also visual observation reveals that SCMA can also correct manual phase assignment errors, if any.

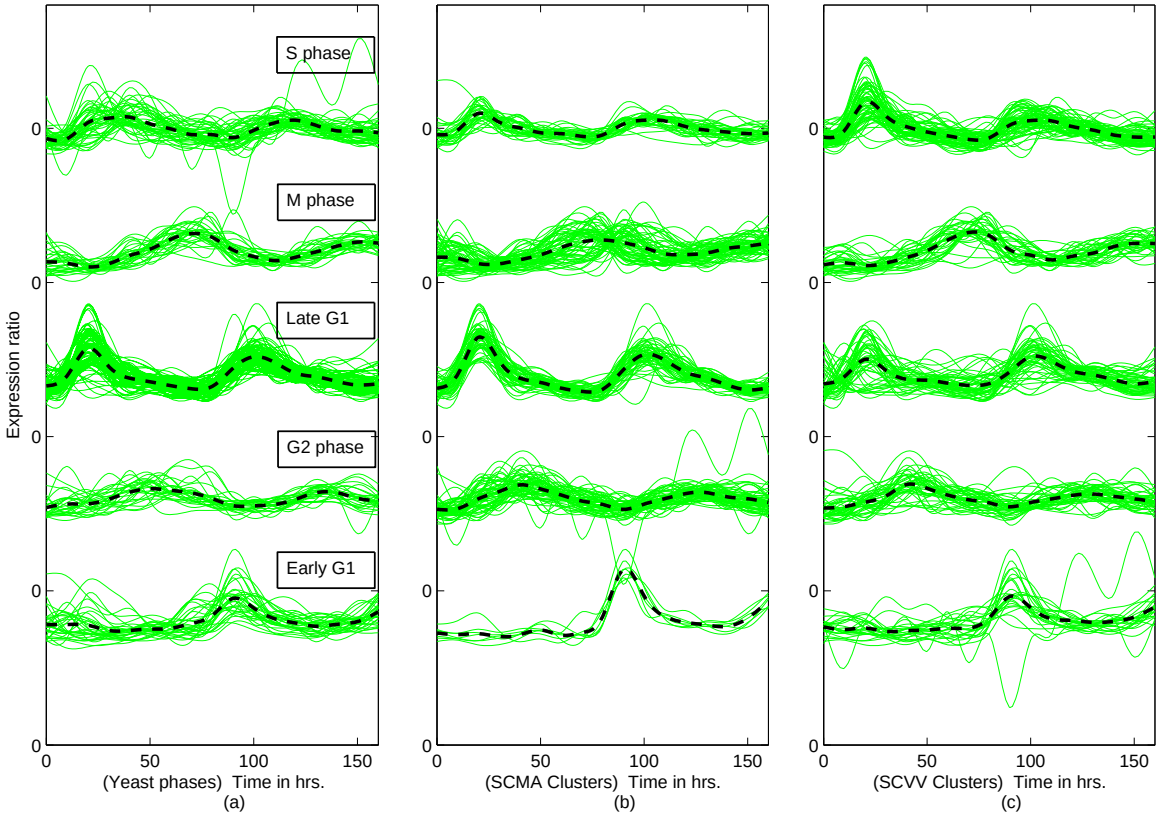


Figure 6.2: (a) Yeast phases [2], (b) SCMA clusters , and (c) SCVV clusters for *Saccharomyces cerevisiae*.

On the other hand, there are noticeable differences between the assignment of genes to clusters by the SCMA and SCVV spectral clustering algorithms. For example, at the peaks of SCVV cluster *C3*, some genes go down, while genes in the corresponding SCMA cluster *C3* evolve very coherently. In order to assess and compare the clustering quantitatively, an

objective measure for comparing the transformation-based spectral clusters with the yeast phases was computed by taking the average classification accuracy, as the number of genes that clustering algorithm *correctly* assigned to one of the phases. Considering each SCMA cluster as a class, $\hat{\mathcal{C}}_{\hat{\mu}_i}$ $(1 \leq i \leq k = 5)$, we trained a c-nearest neighbor (c-NN) classifier with the resulting clusters to classify the data, and evaluated the classification results by following 10-fold cross-validation. By trying different values of c, we found that $c = 7$ is the best for this data set. We used the function given in Equation (6.13) to measure the distance between centroids and the nearest profile(s). This criterion is reasonable, since SCMA and SCVV are unsupervised learning approaches, and hence we do not know the phases beforehand. Thus, the aim is to *"discover"* these phases. In [2], the five phases were determined using biological information, including genomic and phenotypic features observed in the yeast cell cycle experiments. After applying c-NN to the resulting clusters, SCMA and SCVV obtained average classification accuracies of 93.78% and 77.03% respectively (see Table 6.1). Both accuracies are highly considering the fact that SCMA and SCVV are *unsupervised* learning methods. Also, SCMA outperforms SCVV by at least 16% in terms of classification accuracy. In terms of visual observations, it can be inferred that genes from the same cluster generated by SCMA are more compact and separated than in SCVV. Another weakness of SCVV is that for some cases the pattern discovered do not seem to follow the "shapes" of the phases proposed by [2]. As pointed out above, this can also be observed, for example, for the S phase, for which SCVV includes many profiles in SCVV cluster *C1*, while these gene profiles show a high peak at an early stage, and would be better placed in the Late G1 phase.

On the other hand, we applied the S_c index on this data set, in order to determine the best number of clusters. For each $1 \leq K \leq \lceil\sqrt{221}\rceil$, SCMA was run for 20 times and the average value of S_c was obtained. For SCMA, we obtained $K = 10$ as the best number of clusters; $K = 10$ was the best number of clusters for SCVV too. Figure 6.3 shows pairs of similar clusters found after applying the Hungarian algorithm. As in the clustering for the five phases (Figure 6.2), SCVV shows some weaknesses with respect to SCMA. It can be noticed that different clusters are well separated by a number of peaks and the time in which the peaks appear, and hence the "peaks" detected by SCVV do not seem to follow the same trend as those of SCMA. In terms of classification power, in these experiments, SCMA yielded an accuracy of 87.13% and SCVV yielded an accuracy of 57.35% (see Table 6.1), again, showing the superiority of SCMA over SCVV.

The clustering methods were also compared for the data set containing the cell cycle progressions of the fission yeast *Schizosaccharomyces pombe* of [190]. This data set contains

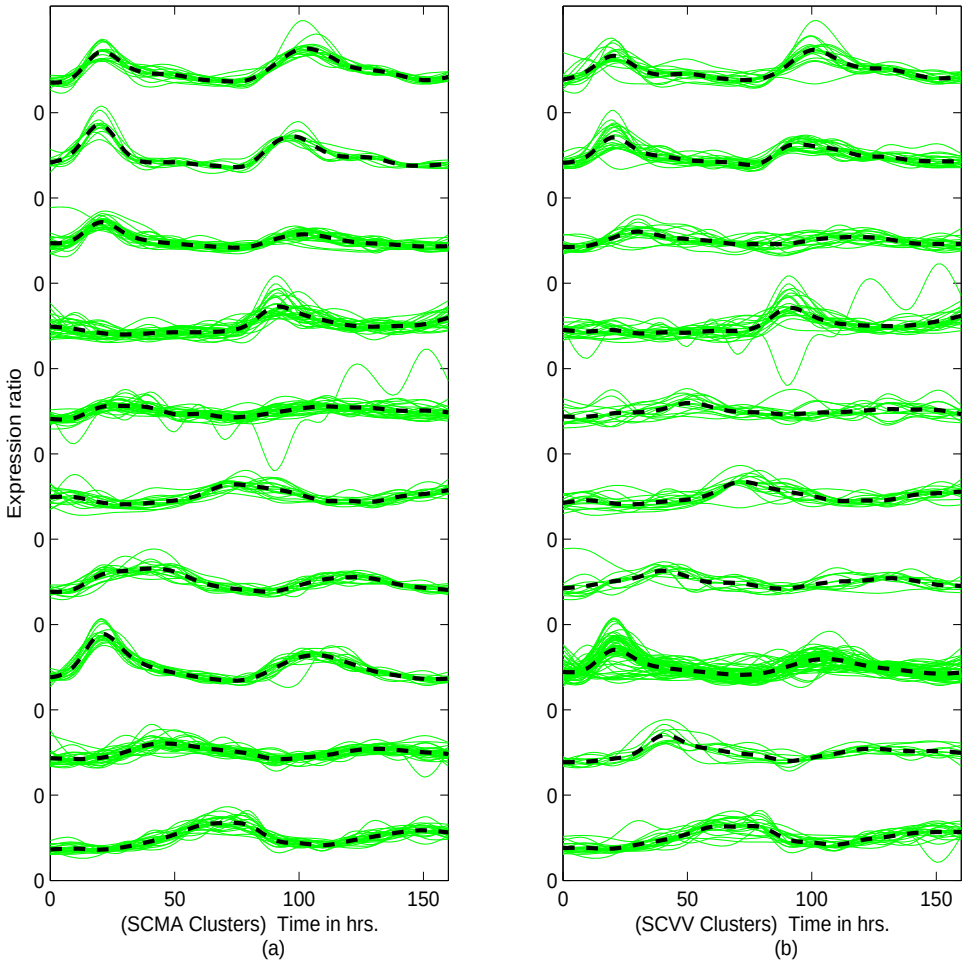


Figure 6.3: (a) SCMA clusters and (b) SCVV clusters for *Saccharomyces cerevisiae* with $K = 10$.

747 genes, containing the expression ratios measured at 14 different time points, for two types of cells, namely, wild-type and cdc25 mutant cells. We compared the performances of SCMA and SCVV methods on the cdc25 type data only. Since this data set does not have class labels, the idea is to discover new clusters on it. For this purpose the SCMA and SCVVA algorithms were applied and the S_c index was used to find the best number of clusters. For both, SCMA and SCVV, we obtained $K = 12$ as the best value of K.

Figure 6.4 shows the clusters obtained by both SCMA and SCVV methods, visualized in same way as those of Figure 6.2 using Hungarian method and multiple alignment, except that in this figure, the assignment of clusters (by means of the Hungarian algorithm) was done by matching clusters from SCMA with those of SCVV. Here too, a visual inspection

shows similarities between the results of the two methods, in general. It can also be observed that the clusters produced by SCMA are more compact and more separated than those obtained by the SCVV. The same objective measure used for the clusters of *Saccharomyces cerevisiae* was also applied to *Schizosaccharomyces pombe*. The alignment-based SCMA obtained a very good average classification accuracy of 88.88%, while SCVV just attained a modest average classification accuracy of 66.91% (see Table 6.1). This demonstrates, again, the superiority of SCMA with respect to SCVV on clustering microarray time-series data.

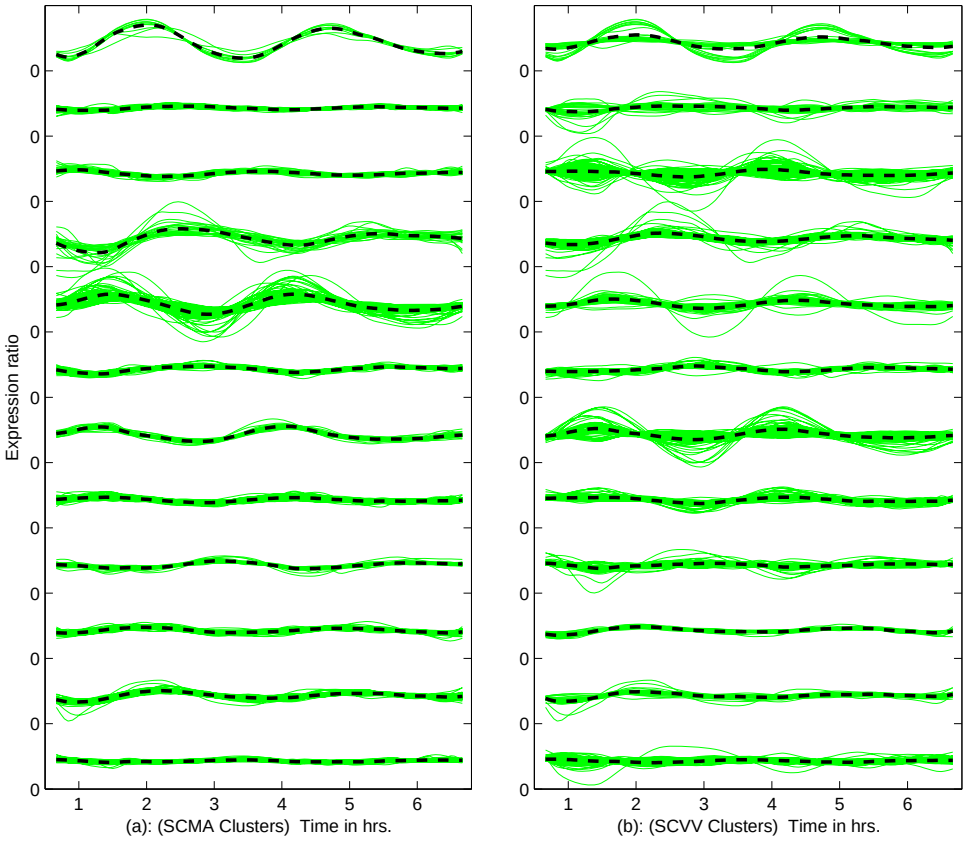


Figure 6.4: (a) SCMA clusters and (b) SCVV clusters with centroids shown, for *Schizosaccharomyces pombe*.

Table 6.1: Accuracies of SCMA and SCVV on two data sets.

Methods	*Saccharomyces cerevisiae*		*Saccharomyces cerevisiae*		*Schizosaccharomyces pombe*	
	Accuracy	k	Accuracy	k	Accuracy	k
SCMA	93.78%	5	87.13%	10	88.88%	12
SCVV	77.03%	5	57.35%	10	66.91%	12

7 Conclusions

In this chapter, we have proposed to cluster microarray time-series data by using spectral clustering combined with two different transformation methods, namely alignment based and variation-based transformations. The spectral clustering algorithm was applied on the data sets obtained by using these two types of transformations. Additionally, a new cluster validity index is proposed for assessing the clustering results of spectral clustering algorithms, due to the nature of these types of algorithms and the unsuitability of the traditional cluster validity indices for this purpose.

We have compared the two transformation methods as another main goal of this chapter, alignment-based transformation versus variation-based transformation, for two well known microarray time-series data sets, namely *Saccharomyces cerevisiae* and *Schizosaccharomyces pombe*. The resulting clustering and an evaluation using the well-known c-nearest neighbor classifier on the clustered data sets demonstrate that the alignment-based transformation substantially outperforms the variation-based transformation. An interesting property to note about the two types of transformations analyzed in this chapter is that one could apply a double-transformation; that is, perform a multiple-alignment on the initial data set, transform the aligned data into variation vectors, and then proceed to cluster the resulting data. However, this, indeed, is not needed in practice, since applying the variation-based transformation to the original data and the resulting data from the alignment-based transformation would produce the same clustering results. This is as a result of the alignment-based transformation being invariant to the variations between time points.

We clarify that the transform mainly contributes to the improvement. This has been empirically proven in [192], where the clustering method was fixed, and different transform methods are compared. However, we believe spectral clustering also contributes to the performance, because it often outperforms many traditional clustering methods. This can be tested by fixing a transform method, and compared spectral clustering with other clustering methods.

Finally, a few words about the potentials of the results presented here for future work is not out of place. One of the issues that is worth investigating is the use of other measures for the cluster validity of spectral clustering. Other graph measurements, including cluster diameter, distance between graphs, among others, can be investigated. The other issue to be investigated is to observe the power of the two transformation schemes combined with spectral clustering on other data sets available in the literature.

Publications

1. The time-series spectral clustering method was preliminary presented in [*Y. Li, N. Subhani, A. Ngom, and L. Rueda, "Alignment-based versus variation based transformation methods for clustering microarray time-series data," ACM International Conference On Bioinformatics and Computational Biology (BCB), Niagara Falls, NY, Aug. 2010, pp.53-61.*].

Chapter 7

High-Order Dynamic Bayesian Network Approaches for Identifying Gene Regulatory Networks

1 Introduction

Accurate and fast reconstruction of *gene regulatory networks* (GRNs) is an important task that has recently become possible due to large-scale high-throughput experiments such as microarray experiments [193]. Gene expression levels obtained over sufficiently large number of time-points can be used to identify GRNs. It is well known that the expressions of a given gene can affect how certain genes are expressed, either down-regulated or up-regulated. GRN represents such causal relationships among genes, encoding all the temporal dependencies between genes in an organism [194]. Regulatory events within an organism are asynchronous, that is, different genes can regulate other genes at different time-scales and with different delays.

Accurate and efficient reconstruction of GRNs from expression time-series data is a computationally hard task, in particular due to the fact that expression levels are measured for a large number of genes numbering in the thousands, and over few number of time-points numbering in the tens. GRN identification methods based on *ordinary differential equations* (ODEs) techniques [194] will be prohibitively slow on such amount of data. GRN inference techniques such as Boolean network [194] methods are not causal and are not

very robust to noise and uncertainty in the data. GRN reconstruction approaches based on *probabilistic graphical models*(PGMs), such as Bayesian networks and Markov random fields [194], have become more popular due to their inherent ability to process uncertain data and their robustness to noise; missing data can also be taken care. PGMs are also more efficient for processing a large number of genes [195].

Bayesian network (BN) is a PGM which compactly represents a joint probability distribution among a set of variables [196]. BNs are directed acyclic graphs (DAGs) which can appropriately model GRNs, that is they model genes as nodes and causal dependencies between genes as edges [197]. Due to their acyclicity constraint, BNs are unable to model self-regulations, feedback loops, and time-delayed interactions, which are the characteristic of GRNs. *Dynamic BNs* (DBNs) are proposed to tackle these limitations by unrolling a BN over time [198]. In DBNs, a transition network between any two consecutive time-points characterizes the GRN; that is only genes at time-point t-1 are supposed to regulate genes at time-points t. This is a first-order assumption allowing to model temporal causal dependencies among genes. *First-order DBNs* (FO-DBNs) , however, cannot model time-delayed interactions longer than one time step. To this effect, *high-order DBNs* (HO-DBNs) were introduced by [199] to model longer time-delayed interactions.

Contributions: In this chapter, we apply PGMs on microarray gene expression time-series data to reconstruct the GRNs. We have the following contributions:

1. We propose the *max-min high-order dynamic Bayesian network* (MMHO-DBN) learning algorithm, in order to reconstruct *time-delayed* gene regulatory networks.

2. We apply the *qualitative probabilistic networks* (QPNs), after obtaining a DBN, to interpret the interactions learned using the concepts of influence and synergy.

3. We have implemented the MMHO-DBN and QPNs in MATLAB, and published it online.

In the rest of this chapter, we first survey the techniques of Bayesian networks in Section 2, we believe that this will help the readers to understand its extensions to HO-DBN and QPN very well. Then in Section 3, we formulate HO-DBN based on its r-order Markov dependency and stationary assumptions. In Section 3.4, we contribute a new HO-DBN structure learning algorithm, called MMHO-DBN, based on an appropriate extension of the original *max-min hill climbing* (MMHC) algorithm of [196] which was devised to alleviate the limitations of the current BN approaches for learning the structure of BNs from static

data. After that, we review the theories of *qualitative probabilistic networks* (QPN) and apply the concepts of influences and synergies to predict different types of interactions in Section 4. The implementation of the above methods are summarized in Section 5. Finally, we investigate the performance of our MMHO-DBN with post-analysis using QPN in Section 6.

2 Bayesian Network

2.1 Concepts

Bayesian network is a probabilistic graphical model representing a joint probability distribution [200]. The cascade decomposition of joint probability distribution can be formulated as

$$\begin{aligned} p(X_1, X_2, \cdots, X_m) &= p(X_1)p(X_2|X_1)p(X_3|X_1, X_2)\cdots p(X_m|X_1, \cdots, X_{m-1}) \\ &= \prod_{i=1}^{m} p(X_i|\mathbf{\Pi}(X_i)), \end{aligned} \tag{7.1}$$

where $\mathbf{\Pi}(X_i)$, or $\mathbf{\Pi}_i$, is the set of variables X_i depends on. These variables are called *parents* of X_i. A Bayesian network can be formulated by two elements: $B = \{G, \boldsymbol{\theta}\}$, where G is the model structure representing the dependency relationship, and $\boldsymbol{\theta}$ is the model parameter, that is the *conditional probability distributions* (CPDs). For discrete variables, the CPDs are *conditional probability tables* (CPTs). We give an example in Figure 7.1, where G represents the decomposition of the joint probability distribution $p(A, B, C, D) = p(A)p(B)p(C|A, B)p(D|C)$, and $\boldsymbol{\theta}$ consists of the (conditional) probability tables.

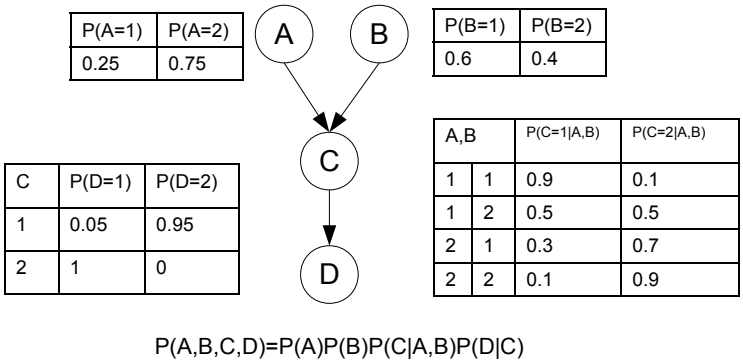


Figure 7.1: An example of a Bayesian network representing a joint probability distribution.

2.2 Learning the Model Structure

The structure G can be reconstructed by learning from data, prior knowledge, or the combination of both [201]. The learning phase can be separated in two steps, that is 1) one obtains the structure, and 2) then learns the parameters. Alternatively, these two steps can be intermingled in an algorithm. Learning the structure G is a model selection problem. According to the existence of hidden variables, we can divide the learning into two cases. First, if there are hidden variables, and only part of the variables are observable, the learning is very difficult. The learning methods are usually based on the *expectation maximization* (EM) [202]. Second, if all the variables are observable, the learning is easier. In our current research, we only focus on learning the model structure from complete data. In our application of learning structure from gene expression time-series data, if there are some missing values, we can estimate the missing values prior to learning a model structure. It is more efficient than learning a model using an EM strategy from data with missing values. For a full treatment of missing values, please see Chapter 8.

2.3 Learning from Complete Data

Now, we derive the learning of the model structure from a Bayesian perspective. The Bayesian model selection can be stated in the following. Given a training set $\boldsymbol{D}_{m\times n}$ containing instances (in columns) of multivariate random variable $\boldsymbol{X}$, the task is to select a structure that maximizes the posterior probability. That is $G = \arg\max p(G|\boldsymbol{D})$.

The posterior can be formulated by the Bayesian theorem:

$$p(G|\boldsymbol{D}) = \frac{p(\boldsymbol{D}|G)p(G)}{p(\boldsymbol{D})}, \tag{7.2}$$

where $p(\boldsymbol{D}) = \sum_{G_i} p(\boldsymbol{D}|G_i)P(G_i)$ is a constant and is independent of the structure G, $p(G)$ is the prior probability on the structure, and $p(D|G)$ is the marginal likelihood. If all structure are treated equally, the model prior can be uniform. If sparse structure is preferred for computational or application reasons, we can use the Gibbs prior as introduced in Section 2.4. Given a model structure, the marginal likelihood is the integration or summation over all possible model parameters:

$$p(\boldsymbol{D}|G) = \int_{\boldsymbol{\theta}} p(\boldsymbol{D}|G,\boldsymbol{\theta})p(\boldsymbol{\theta}|G)d\boldsymbol{\theta}, \tag{7.3}$$

where $\boldsymbol{\theta}$ is a possible parameter setting.

The difficulty of Bayesian learning lies in the computation of the marginal likelihood, as

the integration is intractable. There are two methods to deal with it. The first method is to have an exact closed-form solution, given a proper (conjugate) prior on the parameter. The second is to approximate the marginal likelihood. The *Bayesian Dirichlet equivalence* (BDe) metric belongs to the first method, and the *Bayesian information criterion* (BIC) is an approximate method. We introduce both methods below.

BDe: Exact Closed-Form Solution

Given the model structure and parameter, we assume that the likelihood $p(\boldsymbol{D}|G,\boldsymbol{\theta})$ follows a multinominal distribution, and assume the prior on the parameter follows a Dirichlet distribution, then we can obtain a closed-form solution to the marginal likelihood [201, 203].

The likelihood $p(\boldsymbol{D}|G,\boldsymbol{\theta})$ is decomposable, because if we follow the same procedure as proving the decomposability of $p(X_1, X_2, \cdots, X_m) = \prod_{i=1}^{m} p(X_i|\boldsymbol{\Pi}_i)$ using chain rule of probability, then we have

$$\begin{aligned}
p(\boldsymbol{D}|G,\boldsymbol{\theta}) &= p(\boldsymbol{D}_{X_1}, \boldsymbol{D}_{X_2}, \cdots, \boldsymbol{D}_{X_m}|G,\boldsymbol{\theta}) \\
&= \prod_{i=1}^{m} p(\boldsymbol{D}_{X_i}|\boldsymbol{D}_{\boldsymbol{\Pi}_i}, \boldsymbol{\theta}_i) \\
&= \prod_{i=1}^{m} p(\boldsymbol{D}_{X_i,1}|\boldsymbol{D}_{\boldsymbol{\Pi}_i,1}, \boldsymbol{\theta}_i) \cdots p(\boldsymbol{D}_{X_i,n}|\boldsymbol{D}_{\boldsymbol{\Pi}_i,n}, \boldsymbol{\theta}_i) \\
&= \prod_{i=1}^{m} \prod_{j=1}^{q_i} p(X_i = val(X_i)_1|\boldsymbol{\Pi}_i = val(\boldsymbol{\Pi}_i)_j, \theta_{ij1})^{N_{ij1}} \\
&\quad \cdots p(X_i = val(X_i)_{r_i}|\boldsymbol{\Pi}_i = val(\Pi_i)_j, \theta_{ijr_i})^{N_{ijr_i}} \\
&= \prod_{i=1}^{m} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} p(X_i = val(X_i)_k|\boldsymbol{\Pi}_i = val(\boldsymbol{\Pi}_i)_j, \theta_{ijk})^{N_{ijk}} \\
&= \prod_{i=1}^{m} \prod_{j=1}^{q_i} \prod_{k=1}^{r_i} \theta_{ijk}^{N_{ijk}},
\end{aligned} \tag{7.4}$$

where $val(X_i)$ denotes the set of discrete states of variable X_i,
$N_{ijk} = \sum_{l=1}^{n} I\Big([val(X_i)_k; val(\boldsymbol{\Pi}_i)_j], D_{[X_i;\boldsymbol{\Pi}_i],l}\Big)$, where $I(x,y)$ is an indicator function defined as $I(x,y) = 1$ if x is identical to y, 0 otherwise. In another word, N_{ijk} is the total number of observations where $X_i = val(X_i)_k$ and $\boldsymbol{\Pi}_i = val(\boldsymbol{\Pi}_i)_j$.

We assume the parameter priors of different nodes (that is the (conditional) probability table of each node) are independent, and in a CPT, the parameters under different conditions

are independent as well, then we can decompose the parameter prior as

$$p(\boldsymbol{\theta}) = \sum_{i=1}^{m}\sum_{j=1}^{q_i}\boldsymbol{\theta}_{ij}. \tag{7.5}$$

Having known the formulations of the likelihood (in Equation (7.4)) and prior on parameter (in Equation (7.5)), we can compute the marginal likelihood as below:

$$\begin{aligned}
p(\boldsymbol{D}|G) &= \int_{\theta} p(\boldsymbol{D},\boldsymbol{\theta}|G)d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}} p(\boldsymbol{D}|G,\boldsymbol{\theta})p(\boldsymbol{\theta}|G)d\boldsymbol{\theta} \\
&= \int_{\boldsymbol{\theta}} \prod_{i=1}^{m}\prod_{j=1}^{q_i}\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}} \prod_{i=1}^{m}\prod_{j=1}^{q_i} p(\boldsymbol{\theta}_{ij})d\boldsymbol{\theta} \\
&= \prod_{i=1}^{m}\prod_{j=1}^{q_i}\int_{\boldsymbol{\theta}} \prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}} p(\boldsymbol{\theta}_{ij})d\boldsymbol{\theta}_{ij} \\
&= \prod_{i=1}^{m}\prod_{j=1}^{q_i}\int_{\boldsymbol{\theta}} (\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}}) p(\boldsymbol{\theta}_{ij})d\boldsymbol{\theta}_{ij} \\
&= \prod_{i=1}^{m}\prod_{j=1}^{q_i} E_{p(\boldsymbol{\theta}_{ij}|\boldsymbol{\alpha}_{ij})}(\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}}) \\
&= \prod_{i=1}^{m}\prod_{j=1}^{q_i}\frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})},
\end{aligned} \tag{7.6}$$

where $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$ and $\alpha_{ij} = \sum_{k=1}^{r_i}\alpha_{ijk}$. Please note that the boldface $\boldsymbol{\alpha}_{ij} = [\alpha_{ij1};\cdots;\alpha_{ijr_i}]$ is a vector. In the sixth line of Equation (7.6), $E_{p(\boldsymbol{\theta}_{ij}|\boldsymbol{\alpha}_{ij})}(\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}})$ means the expectation of $\prod_{k=1}^{r_i}\theta_{ijk}^{N_{ijk}}$ with respect to the distribution $p(\boldsymbol{\theta}_{ij}||\boldsymbol{\alpha}_{ij})$. We use the following property to obtain the last line of Equation (7.6). Suppose the multivariate variable $\boldsymbol{Y}$ of length k follows Dirichlet distribution

$$\begin{aligned}
p(\boldsymbol{y}) = Dir(\boldsymbol{y}|\boldsymbol{\alpha}) &= Dir(y_1,\cdots,y_k|\alpha_1,\cdots,\alpha_k) \\
&= \begin{cases} \frac{\prod_{i=1}^{k} y_i^{a_i-1}}{B(\boldsymbol{\alpha})} & \text{if } \forall i: y_i \in (0,1) \text{ and } \sum_{i=1}^{k} y_i = 1 \\ 0 & \text{otherwise} \end{cases},
\end{aligned} \tag{7.7}$$

where

$$B(\boldsymbol{\alpha}) = \frac{\prod_{i=1}^{k}\Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i)}. \tag{7.8}$$

Then, the expectation of $\prod_{i=1}^{k} y_i^{r_i}$ (where $r_i \geq 0$) with respect to $p(\boldsymbol{y})$ can be computed as

$$\begin{aligned} E_{p(\boldsymbol{y})}(\prod_{i=1}^{k} y_i^{r_i}) &= \int \prod_{i=1}^{k} y_i^{r_i} p(\boldsymbol{y}) d\boldsymbol{y} \\ &= \frac{B(\boldsymbol{\alpha}+\boldsymbol{r})}{B(\boldsymbol{\alpha})} \\ &= \frac{\prod_{i=1}^{k}\Gamma(\alpha_i+r_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i+r_i)} \frac{\Gamma(\sum_{i=1}^{k}\alpha_i)}{\prod_{i=1}^{k}\Gamma(\alpha_i)} \\ &= \frac{\Gamma(\sum_{i=1}^{k}\alpha_i)}{\Gamma(\sum_{i=1}^{k}\alpha_i+r_i)} \frac{\prod_{i=1}^{k}\Gamma(\alpha_i+r_i)}{\prod_{i=1}^{k}\Gamma(\alpha_i)}. \end{aligned} \tag{7.9}$$

In Equation (7.6), if the super-parameter α_{ijk} is given by $\alpha_{ijk} = \alpha p_0(X_i = val(X_i)_k, \Pi_i = val(\boldsymbol{\Pi}_i)_j)$ (where α is called *equivalent sample size* (ESS), and $p_0(X_i, \boldsymbol{\Pi}_i)$ is a joint probability distribution of X_i and its parents), then the marginal likelihood is called *Bayesian Dirichlet likelihood equivalent* (BDe) score. If $p_0(X_i, \boldsymbol{\Pi}_i)$ is an uniform distribution, that is $p_0(X_i, \boldsymbol{\Pi}_i) = \frac{1}{|val(X_i)|\cdot|val(\boldsymbol{\Pi}_i)|} = c_0$ (where $|val(X_i)|$ is the size of set $val(X_i)$), we can obtain that $\alpha_{ijk} = \alpha \cdot c_0$. In this case, the marginal likelihood is called BDeu score, where "u" stands for uniform joint probability.

By taking logarithm on the marginal likelihood, as below:

$$\begin{aligned} BDe(\boldsymbol{D}, G) &= \log p(\boldsymbol{D}|G) \\ &= \sum_{i=1}^{m}\sum_{j=1}^{q_i} \big(\log \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})} + \sum_{k=1}^{k} \log \frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})} \big) \\ &= \sum_{i=1}^{m} \Big(\sum_{j=1}^{q_i} \big(\log \frac{\Gamma(\alpha_{ij})}{\Gamma(\alpha_{ij}+N_{ij})} + \sum_{k=1}^{k} \log \frac{\Gamma(\alpha_{ijk}+N_{ijk})}{\Gamma(\alpha_{ijk})} \big) \Big), \end{aligned} \tag{7.10}$$

we can see that the BDe score is decomposable. The decomposability allows to easily score a structure. When a slight change is made based on the previous structure, we update only the local scores associated with the change.

If a variable, say X_i, has no parent, its likelihood is reduced to

$$p(\boldsymbol{D}_i|G_i, \boldsymbol{\theta}_i) = \prod_{k=1}^{r_i} \theta_{ik}^{ik}. \tag{7.11}$$

Suppose its prior follows the Dirichlet distribution:

$$p(\boldsymbol{\theta}_i|G_i, \boldsymbol{\alpha}_i) = Dir(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_i), \tag{7.12}$$

then, its marginal likelihood is reduced to

$$\begin{aligned}
p(\boldsymbol{D}_i|G_i) &= \int_{\boldsymbol{\theta}_i} p(\boldsymbol{D}_i, \boldsymbol{\theta}_i|G_i) d\boldsymbol{\theta}_i \\
&= \int_{\boldsymbol{\theta}_i} p(\boldsymbol{D}_i|G_i, \boldsymbol{\theta}_i) p(\boldsymbol{\theta}_i|G_i) d\boldsymbol{\theta}_i \\
&= \int_{\boldsymbol{\theta}_i} \prod_{k=1}^{r_i} \theta_{ik}^{N_{ik}} p(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i \\
&= \int_{\boldsymbol{\theta}_i} \prod_{k=1}^{r_i} \theta_{ik}^{N_{ik}} p(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i \\
&= \int_{\boldsymbol{\theta}_i} (\prod_{k=1}^{r_i} \theta_{ik}^{N_{ik}}) p(\boldsymbol{\theta}_i) d\boldsymbol{\theta}_i \\
&= E_{p(\boldsymbol{\theta}_i|\boldsymbol{\alpha}_i)} (\prod_{k=1}^{r_i} \theta_{ik}^{N_{ik}}) \\
&= \frac{\Gamma(\alpha_i)}{\Gamma(\alpha_i + N_i)} \prod_{k=1}^{r_i} \frac{\Gamma(\alpha_{ik} + N_{ik})}{\Gamma(\alpha_{ik})}.
\end{aligned} \tag{7.13}$$

BIC: Approximate Solution

By assuming the parameter prior is uniform, we can approximate the parameter $\boldsymbol{\theta}$ by its *maximum likelihood* (ML) estimation $\hat{\boldsymbol{\theta}}$. The marginal likelihood can be approximated by the following equation:

$$\begin{aligned}
\log p(\boldsymbol{D}|G) &\approx \log p(\boldsymbol{D}|G, \hat{\boldsymbol{\theta}}) - \frac{\log n}{2} d(G) \\
&= \log p(\boldsymbol{D}|G, \hat{\boldsymbol{\theta}}) + \log \frac{1}{n^{\frac{d(G)}{2}}},
\end{aligned} \tag{7.14}$$

where $p(\boldsymbol{D}|G, \hat{\boldsymbol{\theta}})$ can be computed by Equation (7.4), $\hat{\boldsymbol{\theta}}$ is the ML estimation of the model parameter given G and $\boldsymbol{D}$, n is the number of training samples, and $\log \frac{1}{n^{\frac{d(G)}{2}}}$ is a penalization term. The more complex the model is, the higher penalty this term enforces. $d(G)$ is the degree of freedom of model G. For complete data, $d(G)$ is the number of free parameters of G (need to consider the sum to one property for discrete case). If there are zero

parameters, $d(G)$ should be further deducted by the number of zero parameters. Therefore, the degree of freedom of a node (say X_i) conditioned by its parents (say $\mathbf{\Pi}_i$) is defined as below:

$$d(G_i) = q(\mathbf{\Pi}_i)(|val(X_i)| - 1) - z(\boldsymbol{\theta}_i), \tag{7.15}$$

where $|val(X_i)|$ denotes the number of discrete states of variable X_i, $q(\mathbf{\Pi}_i) = q_i$ returns the total number of states of the parents, that is

$$q(\mathbf{\Pi}_i) = \begin{cases} 1 & \text{if } \mathbf{\Pi}_i = \emptyset \\ \prod_{j=1}^{|\mathbf{\Pi}_i|} |val(\Pi_{ij})| & \text{otherwise} \end{cases}, \tag{7.16}$$

where $|\mathbf{\Pi}_i|$ denotes the number of parents in set $\mathbf{\Pi}_i$, and $z(\boldsymbol{\theta}_i)$ returns the number of zero parameters in $\boldsymbol{\theta}_i$. We call the right hand side of Equation (7.14) as *Bayesian information criterion* (BIC) score [204]. BIC makes use of the asymptotic behavior which means that, given a large data $\boldsymbol{D}$, the posterior $p(G|\boldsymbol{D})$ is insensitive to the choice of prior under the assumption that the prior of any event is not zero. For a small number of observations, there may be some zeros in the estimated parameter, which will affect the precision of the estimation and degree of freedom, therefore BIC may not work well.

We can easily see that the BIC score is decomposable. This is because

$$BIC(G, \boldsymbol{D}) = \sum_{i=1}^{m} \sum_{j=1}^{q_i} \sum_{k=1}^{r_i} N_{ijk} \log \hat{\theta}_{ijk} - \frac{\log n}{2} d(G_i). \tag{7.17}$$

If the variable X_i has no parent, its BIC score is reduced to

$$BIC(G_i, \boldsymbol{D}_i) = \sum_{k=1}^{r_i} N_{ik} \log \hat{\boldsymbol{\theta}}_{ik} - \frac{\log n}{2} \sum_{i=1}^{m} d(G_i). \tag{7.18}$$

Example of Computing BDeu Score

Now we give an example of how to compute the BDeu score. A working static data set is given in Table 7.1. There are five variables and 12 observations. We wish to compute the BDeu score of the structure as shown in in Figure 7.2. Suppose the equivalent sample size $\alpha = 8$.

According to the structure and observations, we can obtain the frequency table of each variable, as given in Table 7.2. In each cell of a table, the count is given in the parenthesis. The ratio in a cell is the ML estimation of the corresponding parameters.

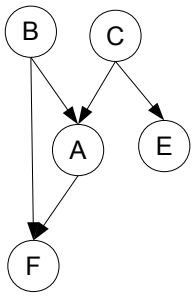


Figure 7.2: A Bayesian network, of which the BDeu and BIC scores are computed.

Table 7.1: Example data.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}	D_{11}	D_{12}
A	2	2	2	1	1	1	2	2	1	2	2	2
B	2	2	2	2	1	1	1	2	2	2	1	2
C	1	2	1	2	2	2	2	2	2	1	1	2
E	2	1	1	1	1	2	2	1	1	2	1	2
F	2	2	2	2	1	2	1	2	2	2	2	1

Table 7.2: Frequency tables and (conditional) probability tables.

(a) Frequency table and CPT of $p(A|B,C)$.

B,C	$A=1$	$A=2$
1,1(1)	0(0)	1(1)
1,2(3)	$\frac{2}{3}$(2)	$\frac{1}{3}$(1)
2,1(3)	0(0)	1(3)
2,2(5)	$\frac{2}{5}$(2)	$\frac{3}{5}$(3)

(b) Frequency table and CPT of $p(F|A,B)$.

A,B	$F=1$	$F=2$
1,1(2)	$\frac{1}{2}$(1)	$\frac{1}{2}$(1)
1,2(2)	0(0)	1(2)
2,1(2)	$\frac{1}{2}$(1)	$\frac{1}{2}$(1)
2,2(6)	$\frac{1}{6}$(1)	$\frac{5}{6}$(5)

(c) Frequency table and PT of $p(B)$.

$B=1$	$B=2$
$\frac{1}{3}$(4)	$\frac{2}{3}$(8)

(d) Frequency table and PT of $p(C)$.

$C=1$	$C=2$
$\frac{1}{3}$(4)	$\frac{2}{3}$(8)

(e) Frequency table and CPT of $p(E|C)$.

C	$E=1$	$E=2$
1(4)	$\frac{1}{2}$(2)	$\frac{1}{2}$(2)
2(8)	$\frac{5}{8}$(5)	$\frac{3}{8}$(3)

The BDeu score of variable A is

$$\begin{aligned} BDeu(G_A, \boldsymbol{D}_A) &= \log\frac{\Gamma(2)}{\Gamma(2+1)} + \log\frac{\Gamma(1+0)}{\Gamma(1)} + \log\frac{\Gamma(1+1)}{\Gamma(1)} \\ &+ \log\frac{\Gamma(2)}{\Gamma(2+3)} + \log\frac{\Gamma(1+2)}{\Gamma(1)} + \log\frac{\Gamma(1+1)}{\Gamma(1)} \\ &+ \log\frac{\Gamma(2)}{\Gamma(2+3)} + \log\frac{\Gamma(1+0)}{\Gamma(1)} + \log\frac{\Gamma(1+3)}{\Gamma(1)} \\ &+ \log\frac{\Gamma(2)}{\Gamma(2+5)} + \log\frac{\Gamma(1+2)}{\Gamma(1)} + \log\frac{\Gamma(1+3)}{\Gamma(1)} \\ &= -4.0943. \end{aligned} \tag{7.19}$$

Similarly, we can compute the BDeu score of the remaining variables: $BDeu(G_B, \boldsymbol{D}_B) = 0.6064$, $BDeu(G_C, \boldsymbol{D}_C) = 0.6064$, $BDeu(G_E, \boldsymbol{D}_E) = -2.3697$, $BDeu(G_F, \boldsymbol{D}_F) = -3.6323$. Therefore, the overall BDeu score of the structure G is -8.8836.

Example of Computing BIC Score

Now we show how to compute the BIC score of a structure. First of all, we can use ML to estimate the CPT of each node as shown in Table 7.2. The degree of freedom is the number of free parameters in each table. If there are zeros in a table, the degree of freedom should be reduced by the number of zeros. For example in Table 7.2(a), the degree of freedom is 4-2=2.

Assuming the training instances are independent, the likelihood can be computed as

$$\begin{aligned}
&\log p(\boldsymbol{D}_{ABC}|\hat{\boldsymbol{\theta}}_A) \\
&= 0\log p(A=1|BC=11,\hat{\boldsymbol{\theta}}_A) + 1\log p(A=2|BC=11,\hat{\boldsymbol{\theta}}_A) \\
&+ 2\log p(A=1|BC=12,\hat{\boldsymbol{\theta}}_A) + 1\log p(A=2|BC=12,\hat{\boldsymbol{\theta}}_A) \\
&+ 0\log p(A=1|BC=21,\hat{\boldsymbol{\theta}}_A) + 3\log p(A=2|BC=21,\hat{\boldsymbol{\theta}}_A) \\
&+ 2\log p(A=1|BC=22,\hat{\boldsymbol{\theta}}_A) + 3\log p(A=2|BC=22,\hat{\boldsymbol{\theta}}_A) \\
&= \log 1 + 2\log\frac{2}{3} + \log\frac{1}{3} + 3\log 1 + 2\log\frac{2}{5} + 3\log\frac{3}{5} \\
&= -5.2746. \qquad (7.20)
\end{aligned}$$

The second term of the BIC score is computed as $2(\log(12)/2) = -2.4849$. Therefore, $BIC(G_A, \boldsymbol{D}_A) = -5.2746 - 2.4849 = -7.7595$. Similarly, we have $BIC(G_B, \boldsymbol{D}_B) = -8.8806$, $BIC(G_C, \boldsymbol{D_C}) = -8.8806$, $BIC(G_E, \boldsymbol{D}_E) = -10.5500$, and $BIC(G_F, \boldsymbol{D}_F) = -9.2033$. Thus, the BIC score of the whole structure is $BIC(G, \boldsymbol{D}) = -45.2741$.

2.4 The Structure Prior

The BDeu and BIC score introduced above do not consider the structure prior, which implies that uniform prior is assumed. In a specific application, informative prior can be used to improve the performance. For example, in the reverse-engineering of GRN, we think a network should be sparse, that is a target gene has only a few number of parents. In this case, we would like to use Gibbs distribution as defined below:

$$p(G|\beta) = \frac{1}{Z}e^{-\beta E(G)}, \qquad (7.21)$$

where $E(G)$ is the total number of edges in G, $\beta \geq 0$ is a parameter, and Z is a normalization term. This term can be added to Equations (7.10) and (7.17).

Furthermore, the structure prior is the key to integrate multiple types of prior knowledge. For example, in the reconstruction of GRN, epigenetic knowledge of histone modifications is combined in the learning of dynamic Bayesian network [205]. In [206], microarray data are combined with biological knowledge, including protein-protein interactions, protein-DNA interactions, binding site information, and existing literature.

2.5 Search Methods

There are an exponential number of feasible DAGs with respect to the number of variables [207]. It has been proven that finding the optimal DAG with respect to the best posterior probability is NP-hard [208]. Therefore, heuristic search methods are applied. For example, computational intelligences, such as genetic algorithm, simulated annealing, and hill climbing algorithms, are widely used. *Markov chain Monte Carlo* (MCMC) method is a sampling algorithm which is favored by statisticians.

While exploring the whole structural space is intractable, we can restrict the search space before conducting the iterations of search and scoring. This is call constrained-and-search strategy [196]. Now, we briefly introduce one of such algorithm which is called *max-min hill climbing* (MMHC) [196], which was originally devised for learning the structure of BNs from static data. We have extended this algorithm for learning high-order dynamic Bayesian network, which will be discussed in latter section.

MMHC was introduced in [196] as a fast, scalable, and reliable BN learning method which overcomes the perceived limitations of the current state-of-the-art BN algorithms and which also exist in current HO-DBN algorithms. MMHC is a hybrid BN method; it first uses *constraint-based Bayesian network learning* [209] to learn the skeleton (i.e. an undirected graph) of a BN, and then performs a *search-and-score Bayesian network learning* on the skeleton in order to orient its edges. It is the skeleton learning phase which gives MMHC its reliability and accuracy, its efficiency, and more importantly, its ability to scale to distributions with thousands of variables. MMHC is also a local learning method which does not require a user to estimate the number of parents for each variable, as it discovers the maximum number of possible *parents and children* (PC), of each variable during the skeleton learning phase. This discovery was proven accurate and more efficient than that of the hybrid *sparse candidate* (SC) algorithm [197] and that of the constraint-based methods such as PC algorithm [209].

In addition to the challenge of exponential structure space, another weakness of discrete

BN is that the number of parameters in a CPT of a node conditioned by its parents increases exponentially as the number of parents increases. For this reason, the number of parents of each node is usually limited. This leads to a sparse BN. Sparse candidate [197] is one of such algorithm.

2.6 Reconstructing Gene Regulatory Networks by BN

Friedman *et al.* [197] did a seminal work of reconstructing GRN via learning Bayesian network from gene expression time-series data, though temporal relations were not taken into account at that time. Instead of learning a single model from the data, the confidence of many features are built on models learned from many data sets generated by bootstrap. This method has two phases. First, instead of using MCMC sampling to generate many networks, the bootstrap method is used to perturb the gene expression data to generate many data sets. For each data set, a BN is learned by the sparse candidate method. Second, the confidence of each features is computed over all networks. A feature is a boolean function that describes a relationship between a pair of nodes. In [197], the Markov and order features are considered. It is stated that very low false positive of the features can be obtained, even though there are only a small number of samples.

Ignoring the temporal relationships would make the structure learning algorithm more efficient. However, without considering the temporal information, BNs are unable to model self-regulations, feedback loops, and time-delayed interactions, which are the characteristic of GRNs. This motivates us to turn to dynamic Bayesian netowrks as discussed in the following sections.

3 High-Order DBNs for Gene Regulatory Network Identification [1]

3.1 Introduction

In the section, we propose a new high-order dynamic Bayesian network (HO-DBN) learning approach, called *Max-Min high-order DBN* (MMHO-DBN), for discrete time-series data. MMHO-DBN explicitly models the time lags between parents and target in an efficient manner. It extends the *Max-Min hill-climbing Bayesian network* (MMHC-BN) technique which was originally devised for learning a BN's structure from static data. Both Max-Min

[1] This section is published in [210]

approaches are hybrid local learning methods which fuse concepts from both constraint-based Bayesian techniques and search-and-score Bayesian methods. The MMHO-DBN first uses constraint-based ideas to limit the space of potential structure and then applies search-and-score ideas to search for an optimal HO-DBN structure. We evaluated the ability of our MMHO-DBN approach to identify *gene regulatory networks* (GRN's) from gene expression time-series data. Preliminary results on artificial and real gene expression time-series are encouraging and show that it is able to learn (long) time-delayed relationships between genes, and faster than current HO-DBN learning methods.

The rest of this section is organized as follows. Section 3.2 presents GRN modeling with HO-DBNs. In Section 3.3, we discuss current methods to learning HO-DBN structures for reconstructing GRNs from microarray time-series data. Then we introduce our MMHO-DBN structure learning method in Section 3.4. Preliminary results and discussions are presented in Section 3.5. Finally, we conclude and suggest possible directions of research in HO-DBN learning.

3.2 Modeling Time-Delayed Regulations with HO-DBNs

Let us consider a gene expression time-series data set $\mathbf{g}_{T\times N} = (\mathbf{g}_1, \ldots, \mathbf{g}_T)^{\mathrm{T}}$ summarizing the observations (i.e., expression levels) of N genes at T time-points. Row vector $\mathbf{g}_t = (g_{t,1}, \ldots, g_{t,N})^{\mathrm{T}}$, contains the expression levels of the N genes measured at time-point t and where $g_{t,j}$ is an observation from the random variable $G_{t,j}$, for $1 \leq t \leq T$ and $1 \leq j \leq N$.

The DBN [211] usually refers to the first-order DBN (FO-DBN). FO-DBN assumes a Markov dependency of order 1 over time; that is, the expression level of a gene at time t depends only on the expression levels of the genes at time $t-1$ and t. FO-DBN is defined by a pair of structures (S_{t-1}, S_t) corresponding to networks at time slices $t-1$ and t, and a transition network $S_{[t-1,t]}$ of interactions between S_{t-1} and S_t; thus, $S_{[t-1,t]}$ has $2N$ nodes. The FO-DBN structure is obtained by unrolling the transition network over time, and the parents of a variable $G_{t,j}$ are from time-slices $t-1$ and t only. FO-DBN cannot model time-delayed interactions more than 1 time unit which occur in GRNs but can be extended to allow higher-order interactions among variables.

High-order DBNs (HO-DBNs) have been proposed to model time-delayed interactions between genes, where the structure and parameters of the HO-DBN are learned by assuming a fixed order $r > 1$ [199, 195, 212, 213] representing the maximum allowed time-delay among genes. The r-order DBN (r-DBN) assumes an r-order Markov dependency over time. It is defined by an $(r+1)$-tuple of structures $(S_{t-r}, S_{t-(r-1)}, \ldots, S_{t-1}, S_t)$ corresponding to networks at time-slices $t-r, \ldots, t$, and a transition network $S_{[t-r,t-(r-1),\ldots,t-1,t]}$ (or $S_{[t-r,t]}$,

for short) representing the causal connectivity structure between each network S_{t-l} and S_t, $1 \leq l \leq r$. The structure of the r-DBN is obtained by unrolling $S_{[t-r,t]}$ over time. The transition network $S_{[t-r,t]}$ consists of $(r+1)N$ nodes, and the parents of a variable $G_{t,j}$ are chosen from the set of variable $\bigcup_{l=0}^{r} \mathbf{G}_{t-l}$, where $\mathbf{G}_i = \{G_{i,1}, G_{i,2}, \ldots, G_{i,N}\}$ is the set of N random variables at time-slice i. We assume an r-order stationary [2] Markov chain and that the networks S_{t-l}, $0 \leq l \leq r$, have no edges. The GRN can be represented as a matrix $\mathbf{C} = \{c_{i,j}\}_{N \times N}$, where $1 \leq c_{i,j} \leq r$ denotes the time delay of regulation between gene i and its parent gene j. In Figure 7.3, we give a comparison of stationary HO-DBN and non-stationary HO-DBN. In Figure 7.4, we give an example of the transition network of a DBN under the second-order stationary Markov assumption. In this figure, A, B, C, and D are called "attributes". They correspond to nodes in the folded transition network. In our application, they are genes in a GRN.

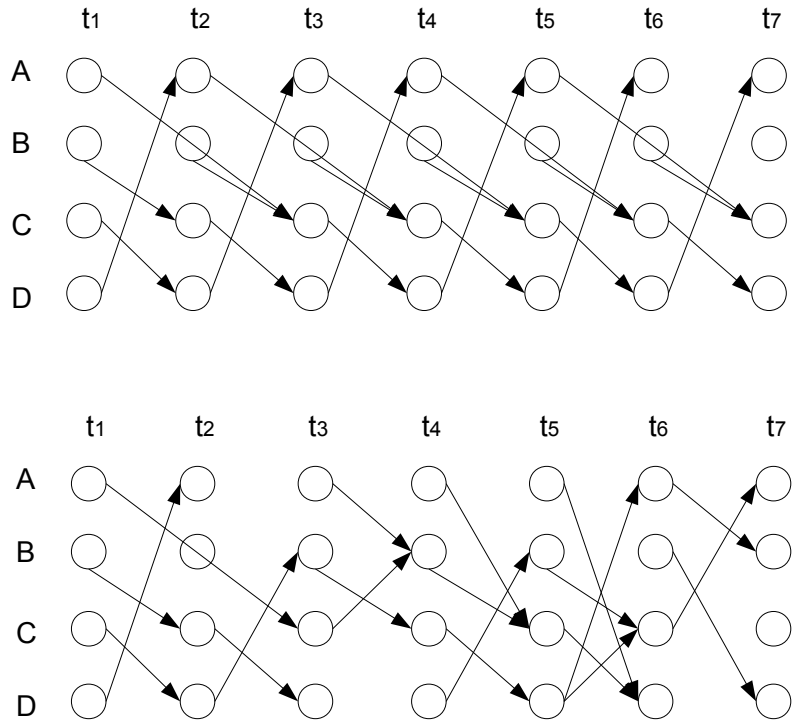


Figure 7.3: An example of a second-order HO-DBN (top) and a non-stationary HO-DBN (bottom).

Let $\mathbf{G} = (\mathbf{G}_1, \ldots, \mathbf{G}_T)^{\mathrm{T}}$ where each $\mathbf{G}_t = (G_{t,1}, \ldots, G_{t,N})^{\mathrm{T}}$ is a N-dimensional random

[2]The stationarity is usually assumed to simplify modeling. It is also determined by specific domain knowledge. Whether a process is stationary or non-stationary can be checked by statistical hypothesis test.

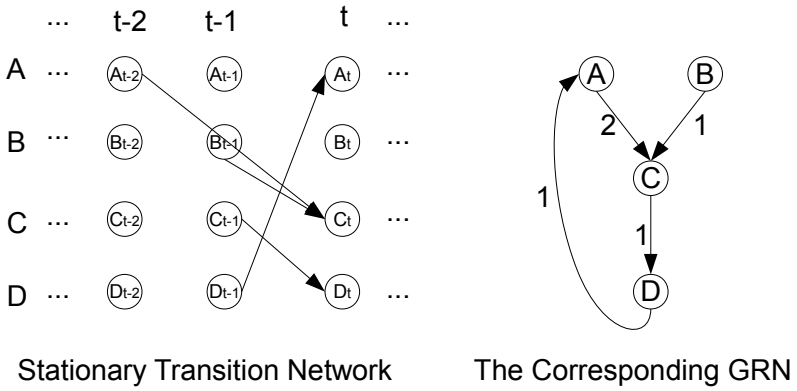


Figure 7.4: An example of a DBN under the second-order stationary Markov assumption. Left: stationary transition network. Right: the GRN obtained by folding the transition network.

variable vector. Under the Bayesian framework, a gene is a random variable and we consider directed acyclic graph S and an r-order stationary Markov assumption between nodes. The r-order DBN (r-DBN) assumes an r-order Markov dependency over time:

$$P(\mathbf{G}_t|\mathbf{G}_{t-1},\ldots,\mathbf{G}_1) = P(\mathbf{G}_t|\mathbf{G}_{t-1},\ldots,\mathbf{G}_{t-r}). \tag{7.22}$$

The r-DBN thus decomposes the joint probability distribution of $\mathbf{G}$ given the structure S into a product of conditional probabilities by assuming independence of non-descendant variables as:

$$P(\mathbf{G}) = \prod_{t=1}^{T} P(\mathbf{G}_t|\mathbf{G}_{t-1},\ldots,\mathbf{G}_{t-r}). \tag{7.23}$$

Let $\mathbf{P}_{t-l,j} = (P_{t-l,j,1},\ldots,P_{t-l,j,q_{t-l,j}})^{\mathrm{T}}$ be the $q_{t-l,j}$-dimensional random vector of the parents of the j-th gene at time $t-l$, $1 \le l \le r$; $\mathbf{P}_{t-l,j} = \emptyset$ if $t-l \le 0$. We define the set of all parents of the j-th gene as the q_j-dimensional vector $\mathbf{P}_{[1,r],j} = \bigcup_{l=1}^{r} \mathbf{P}_{t-l,j}$, where $q_j = \sum_{l=1}^{r} q_{t-l,j}$. Then the conditional probabilities $P(\mathbf{G}_t|\mathbf{G}_{t-1},\ldots,\mathbf{G}_{t-r})$ can be decomposed into a product of conditional probabilities of each gene given its parents $\mathbf{P}_{[1,r],j}$ as:

$$P(\mathbf{G}_t|\mathbf{G}_{t-1},\ldots,\mathbf{G}_{t-r}) = \prod_{j=1}^{N} P(G_{t,j}|\mathbf{P}_{t-1,j}\cup\cdots\cup\mathbf{P}_{t-r,j}) = \prod_{j=1}^{N} P(G_{t,j}|\mathbf{P}_{[1,r],j}). \quad (7.24)$$

The important issue pertaining to modeling GRNs by r-DBNs is how to find the conditional probabilities $P(G_{t,j}|\mathbf{P}_{[1,r],j})$ which best explain the data. To determine the optimal $P(G_{t,j}|\mathbf{P}_{[1,r],j})$, we parameterize $P(\mathbf{G})$ by a parameter vector $\boldsymbol{\theta} = (\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_N)$ and transfer the determination of the optimal $P(\mathbf{G})$ into the estimation of the best θ. Parameterizing and substituting Equation (7.24) into Equation (7.23) we obtain the discrete r-DBN model:

$$P(\mathbf{G}|\boldsymbol{\theta}) = \prod_{t=1}^{T}\prod_{j=1}^{N} P(G_{t,j}|\mathbf{P}_{[1,r],j};\boldsymbol{\theta}_j). \quad (7.25)$$

By using r-DBN models, we can model higher-order GRN interactions from time-series data, when we know the true relationships among the genes. Such relationships are still unknown, and hence, it is necessary to devise criteria for evaluating the goodness of a structure, and then, devise search algorithms for searching the large space of candidate structures. In this space, the optimal structure $\hat{S}$ is the one which maximizes the posterior probability $P(S|\mathbf{G})$. From Bayes theorem we have:

$$P(S|\mathbf{G}) = \frac{P(S)P(\mathbf{G}|S)}{P(\mathbf{G})} \propto P(S)P(\mathbf{G}|S), \quad (7.26)$$

where $P(S)$ is the prior probability of the network structure S and $P(\mathbf{G}) = \sum_S P(S)$ $\int_{\boldsymbol{\theta}} P(\mathbf{G}|S,\boldsymbol{\theta})P(\boldsymbol{\theta}|S)d\boldsymbol{\theta}$ is constant, independent of S, and can be removed since it does not relate to structure evaluation. Given the set of conditional distributions with parameter $\boldsymbol{\theta}$, we can express the marginal likelihood of the time-series data as:

$$P(\mathbf{G}|S) = \int_{\boldsymbol{\theta}} P(\mathbf{G}|S,\boldsymbol{\theta})P(\boldsymbol{\theta}|S)d\boldsymbol{\theta}, \quad (7.27)$$

where $P(\boldsymbol{\theta}|S)$ is the prior probability of the parameter $\boldsymbol{\theta}$ and $P(\mathbf{G}|S,\boldsymbol{\theta}) = P(\mathbf{G}|\boldsymbol{\theta}_S)$; note that we write $\boldsymbol{\theta}_S = (\boldsymbol{\theta}_1,\ldots,\boldsymbol{\theta}_N)$ since the form of $\boldsymbol{\theta}$ in Equation (7.25) is equivalent to the network structure S. The *maximum a-posteriori* (MAP) estimate of the optimal structure $\hat{S}$ is then given as:

$$\hat{S}_{\text{MAP}} = \arg\max_{S} P(S) \int_{\boldsymbol{\theta}_S} P(\mathbf{G}|\boldsymbol{\theta}_S) P(\boldsymbol{\theta}_S|S) d\boldsymbol{\theta}_S. \tag{7.28}$$

The problem which remains now is to 1) determine the conditional probabilities $P(G_{t,j}|\mathbf{P}_{[1,r],j}; \boldsymbol{\theta}_j)$, 2) determine the prior probabilities $P(S)$ and $P(\boldsymbol{\theta}_S|S)$, 3) compute the high-dimensional integral, and 4) search for the optimal $\hat{S}_{\text{MAP}}$. Points 1) through 3) combine into a single criterion for learning GRNs based on r-DBN, and which is used within a search algorithm in point 4) to evaluate the goodness of candidate GRN structures.

For the discrete model we assume that gene expression values are discretized into d levels such that $g_{t,j} \in \{1, \ldots, d\}$ and d denotes the maximum level of expression of any gene. The number of distinct states that $\mathbf{P}_{[1,r],j}$ (the parents set of the j-th gene) can take is $Q_j = d^{q_j}$. Each state of $\mathbf{P}_{[1,r],j}$ is also associated with a lag vector $\mathbf{L}_{[1,r],j} \in \{1, \ldots, r\}^{q_j}$ containing the delay of each parent of gene j; hence, the number of distinct lag vectors for gene j is $L_j = r^{q_j}$. Let $\theta_{j,q,l,k} = P(G_{t,j} = k|\mathbf{P}_{[1,r],j} = q, \mathbf{L}_{[1,r],j} = l)$ and $N_{j,q,l,k} = \sum_{t=1}^{T} \delta(G_{t,j} = k, \mathbf{P}_{[1,r],j} = q, \mathbf{L}_{[1,r],j} = l)$ be the number of observations satisfying $G_{t,j} = k$, $\mathbf{P}_{[1,r],j} = q$, and $\mathbf{L}_{[1,r],j} = l$, and $N_{j,q,l,k}^{[t-r,t]}$ be the number of such observations in the transition network $S_{[t-r,t]}$ for $1 \leq t \leq T$. Using the property of decomposability [198], we can model $P(\mathbf{G}; \boldsymbol{\theta}_S)$ as a multinomial distribution with parameter $\boldsymbol{\theta}_S = (\theta_{j,q,l,k})_{N \times Q_j \times L_j \times d} = (\theta_{1,1,1,1}, \ldots, \theta_{N,Q_j,L_j,d})^{\mathrm{T}}$:

$$\begin{aligned} P(\mathbf{G}|\boldsymbol{\theta}_S) &= \prod_{t=1}^{T} \prod_{j=1}^{N} \prod_{q=1}^{Q_j} \prod_{l=1}^{L_j} \prod_{k=1}^{d} \theta_{j,q,l,k}^{N_{j,q,l,k}^{[t-r,t]}} \\ &= \prod_{j=1}^{N} \prod_{q=1}^{Q_j} \prod_{l=1}^{L_j} \prod_{k=1}^{d} \theta_{j,q,l,k}^{N_{j,q,l,k}} \end{aligned} \tag{7.29}$$

Define $\boldsymbol{\theta}_S = \bigcup_{j=1}^{N} \{\boldsymbol{\theta}_j\}$, $\boldsymbol{\theta}_j = \bigcup_{q=1,l=1}^{Q_j,L_j} \{\boldsymbol{\theta}_{j,q,l}\}$, and $\boldsymbol{\theta}_{j,q,l} = \bigcup_{k=1}^{d} \{\theta_{j,q,l,k}\}$. Assuming that the global and local parameter vectors are independent of each other, then we can decompose the prior distribution on $\boldsymbol{\theta}_S$ as:

$$
\begin{aligned}
P(\boldsymbol{\theta}_S|S) &= \prod_{j=1}^{N} P(\boldsymbol{\theta}_j|S) \\
&= \prod_{j=1}^{N}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j} P(\boldsymbol{\theta}_{j,q,l}|S) \\
&= \prod_{j=1}^{N}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j}\prod_{k=1}^{d} P(\theta_{j,q,l,k}).
\end{aligned} \tag{7.30}
$$

Substituting Equations (7.29) and (7.30) into Equation (7.28), we obtain

$$
P(\mathbf{G}|S) = \prod_{j=1}^{N}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j}\int \prod_{k=1}^{d} \theta_{j,q,l,k}^{N_{j,q,l,k}} P(\theta_{j,q,l,k}) d\theta_{j,q,l,k}. \tag{7.31}
$$

Assuming Dirichlet distribution [214] with hyper-parameters $\boldsymbol{\alpha}_S = \bigcup_{j=1}^{N}\{\boldsymbol{\alpha}_j\}$ and $\boldsymbol{\alpha}_j = \bigcup_{q=1,l=1,k=1}^{Q_j,L_j,d}\{\alpha_{j,q,l,k}\}$ as the prior distribution on the global and local parameters, then:

$$
\begin{aligned}
P(\boldsymbol{\theta}_j|S) &= Dir(\boldsymbol{\theta}_j|\boldsymbol{\alpha}_j) \\
&= \frac{\Gamma(\sum_{q=1}^{Q_j}\sum_{l=1}^{L_j}\sum_{k=1}^{d}\alpha_{j,q,l,k})}{\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j}\prod_{k=1}^{d}\Gamma(\alpha_{j,q,l,k})}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j}\prod_{k=1}^{d}\theta_{j,q,l,k}^{\alpha_{j,q,l,k}-1},
\end{aligned} \tag{7.32}
$$

where $\Gamma(\cdot)$ is the Gamma function [214], which satisfies $\Gamma(x+1) = x\Gamma(x)$ and $\Gamma(1) = 1$. Using the Dirichlet priors of Equation (7.32), the high-dimensional integral in Equation (7.31) is solved to obtain a closed-form formula:

$$
P(\mathbf{G}|S) = \prod_{j=1}^{N}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j}\frac{\Gamma(\alpha_{j,q,l})}{\Gamma(\alpha_{j,q,l}+N_{j,q,l})}\prod_{k=1}^{d}\frac{\Gamma(\alpha_{j,q,l,k}+N_{j,q,l,k})}{\Gamma(\alpha_{j,q,l,k})}, \tag{7.33}
$$

where $N_{j,q,l} = \sum_{k=1}^{d} N_{j,q,l,k}$ and $\alpha_{j,q,l} = \sum_{k=1}^{d}\alpha_{j,q,l,k}$. Equation (7.33) corresponds to the

1. *Bayesian Dirichlet equivalence* (BDe) metric of [201] when $\alpha_{j,q,l,k} = \alpha P(G_{t,j} = k, \mathbf{P}_{[1,r],j} = q, \mathbf{L}_{[1,r],j} = l|S)$, and $\alpha \geq 0$ is the equivalent sample size parameter.

2. *BDe uniform* (BDeu) metric of [215] when $\alpha_{j,q,l,k} = \frac{\alpha}{dL_jQ_j}$.

3. *K2 metric* of [216] when $\alpha_{j,q,l,k} = 1$.

To complete the information required to derive the MAP estimate, $\hat{S}_{\text{MAP}}$, we must consider the prior probability $P(S)$ of a given structure. Let S_j be the local structure at the j-th gene, we can set the prior probability of S, that is:

$$P(S) = e^{-(\#\text{ of edges})} = \prod_{j=1}^{N} P(S_j) = \prod_{j=1}^{N} e^{-q_j}. \tag{7.34}$$

Taking the product of Equations (7.33) and (7.34) yields the r-DBN learning criterion. We note that this criterion is decomposable since it can be written as a product of local scores, each of which is a function of the j-th gene only. That is, we have

$$P(S|\mathbf{G}) \propto P(S)P(\mathbf{G}|S) = \prod_{j=1}^{N}\prod_{q=1}^{Q_j}\prod_{l=1}^{L_j} e^{-q_j}\frac{\Gamma(\alpha_{j,q,l})}{\Gamma(\alpha_{j,q,l}+N_{j,q,l})}\prod_{k=1}^{d}\frac{\Gamma(\alpha_{j,q,l,k}+N_{j,q,l,k})}{\Gamma(\alpha_{j,q,l,k})}. \tag{7.35}$$

If we assume that every structure in the structure space is equally probable a priori, then we can simplify Equation (7.28) to use the *maximum likelihood* (ML) estimate of the optimal structure $\hat{S}$ given as

$$\hat{S}_{\text{ML}} = \arg\max_{S} \int_{\boldsymbol{\theta}_S} P(\mathbf{G}|\boldsymbol{\theta}_S)P(\boldsymbol{\theta}_S|S)d\boldsymbol{\theta}_S. \tag{7.36}$$

In this case we estimate the parameters $\theta_{j,q,l,k}$ as

$$\theta_{j,q,l,k} = \frac{N_{j,q,l,k}}{\sum_{k=1}^{d} N_{j,q,l,k}}, \tag{7.37}$$

and then use the *Bayesian information criterion* (BIC) of [204] to approximates the integral as:

$$BIC = \sum_{j=1}^{N}\sum_{q=1}^{Q_j}\sum_{l=1}^{L_j}\sum_{k=1}^{d} N_{j,q,l,k}\log\frac{N_{j,q,l,k}}{\sum_{k=1}^{d} N_{j,q,l,k}}. \tag{7.38}$$

The BIC score does not require priors over parameters and is computationally faster to compute but less accurate than the BDe, BDeu, and K2 scores. The BIC score is good when given a large data set. Finding the optimal structure of a BNs is known to be NP-hard [207] as the number of structures increases super-exponentially with the number of nodes. The learning problem, hence, becomes much harder as the order r increases.

3.3 Related Works

Few methods have been proposed in literature to address the difficulty of modeling time-delayed regulations. The authors of [217] devised a three-steps approach which essentially transforms the HO-DBN problem into FO-DBN problem thus avoiding to learn from an extremely large space of parameters. Their approach, DBN-ZC, is a local search-and-score learning method which proceeds as follows. First, DBN-ZC limits the potential regulators of each target gene T to only those genes with either earlier or simultaneous expression changes in relation to target T; thus significantly reducing the computational effort in the subsequent structure learning phase. Second, the time difference between the initial gene expression change of a potential regulator and its target T is taken as a fair estimation of the time-delay between the two genes. Third, the time-series profiles of the potential regulators are appropriately aligned to that of T according to their time-lags with T, and then a search-and-score based FO-DBN learning is performed to select the regulators with the highest log-marginal likelihood as the final set of regulators of T. In [199], a two-steps heuristic framework is devised to learn r-DBNs from time-series expression data. First, pairs of variables $G_{t-l,j}, G_{t,k}$ with time lag l, for $1 \leq l \leq r$, $1 \leq t \leq T$ and $1 \leq j, k \leq N$, having mutual information above a given threshold are determined; this step essentially initializes the transition network $S_{[t-r,t]}$. Then in the second step, genetic algorithm (GA) is applied, given the initial transition network $S_{[t-r,t]}$, to find the structure having the highest *maximum likelihood* or the *maximum description length* (MDL) score. Being a population-based optimization method using implicit parallelism, GA is able to search very large spaces given an good representation of the GRN and appropriate genetic operators. Chaturvedi and Rajapakse [213] used prior biological knowledge contained in current protein interaction networks (PINs) as a mean of limiting the search space. The authors modeled time-delayed regulations using a skip-chain model which predicts edges between non-consecutive time-points (called *skip-edges*) based on the prior knowledge in the given PINs. Viterbi approximation of DBNs is used to select the best skip-edges and combined appropriately with the BIC score which selects edges between two consecutive time-points (non skip-edges called *linear-edges* in the paper). In [212], the variable-order DBN (VO-DBN) approach is introduced to automatically find the delays of regulations between genes. In HO-DBNs, the order r (or the maximum delay of regulation) is fixed a priori before learning the structure. In the VO-DBN, however, the optimal order r and the optimal structure of which are learned using a *Markov chain Monte Carlo* (MCMC), which uses an appropriate acceptance mechanism allowing to optimize both order and structure.

3.4 Max-Min High-Order DBNs

In this section, we present an extension of the *Max-Min hill-climbing* (MMHC) heuristic which was originally devised for learning the structure of BNs from static data. MMHC-BN was introduced in [196] as a fast, scalable, and reliable BN learning method which overcomes the perceived limitations of the current state-of-the-art BN algorithms and which also exist in current HO-DBN algorithms. MMHC-BN is a hybrid BN method; it first uses *constraint-based Bayesian learning* [209] to learn the skeleton (i.e. an undirected graph) of a BN, and then performs a *search-and-score Bayesian learning* on the skeleton in order to orient its edges. It is the skeleton learning phase that gives MMHC-BN its reliability and accuracy, its efficiency, and more importantly, its ability to scale to distributions with thousands of variables. MMHC-BN is also a local learning method which does not require the user to estimate the number of parents for each variable as it discovers the maximum number of possible parents and children (PC), of each variable during the skeleton learning phase. This discovery was proven accurate and more efficient than that of the hybrid *sparse candidate* (SC) algorithm [197] and that of the constraint-based methods such as PC algorithm [209].

Our Max-Min High-Order DBN (MMHO-DBN) structure learning method is shown in Algorithm 7.1. The MMHO-DBN approach proceeds with the two MMHC-BN phases, except both phases are appropriately modified to consider a discrete-time stochastic process $\mathbf{G} = (\mathbf{G}_1, \ldots, \mathbf{G}_T)^{\mathrm{T}}$ having a joint probability distribution $\mathcal{P}$ and where each $\mathbf{G}_t = (G_{t,1}, \ldots, G_{t,N})^{\mathrm{T}}$ is a N-dimensional random variable vector taking discrete values.

In Phase 1 of Algorithm 7.1 we modified the local discovery method of [196], the *Max-Min parent and children* (MMPC) algorithm in order to compute the maximum possible parent set, $\mathbf{\Pi}_{[t-r,t],j}$, of each target variable $G_{t,j}$. Given a target variable T and statistical non-stochastic data $\mathcal{D}$, the original MMPC algorithm returns its *maximum possible set of parent and children*, PC(T), provided that the *faithfulness assumption* [209] holds and that the statistical tests performed return reliable result. The faithfulness assumption ensures that the PC(T) set is unique among all BNs faithful to the same distribution; a node may be T's parent in one BN and T's child in another BN, however PC(T) remains the same in both BNs. MMPC is a constraint-based search algorithm which essentially learns the skeleton of a BN; that is, it identifies the existence of edges to and from targets T without identifying the orientation of the edges. The uniqueness of the PC(T) is also true for r-DBNs faithful to the distribution $\mathcal{P}$. Here, however, the edges will be oriented due to the temporal dependencies, and thus we need only find the *maximum possible set of parents* of T; the children of T will be determined following the temporal dependencies. In Algorithm 7.1 above, the *dynamic Max-Min parent* (DMMP) algorithm shown in Algorithm 7.2 is

Algorithm 7.1 The *MMHO-DBN* Algorithm

Input: $\mathbf{g}_{T\times N} = (\mathbf{g}_1, \ldots, \mathbf{g}_T)^{\mathrm{T}}$: time-series data
r: maximum time delay
α: significance level
φ: cardinality limit for $\mathbf{P}_{[t-r,t],j}$
γ: cardinality limit for exhaustive search
Output: $\hat{S}$: best DAG on the variables in $\mathbf{g}_{T\times N}$

{**Phase 1**: Restrict candidate parents}
for every variable $G_{t,j} \in \mathbf{G}$ **do**
$\mathbf{\Pi}_{[t-r,t],j} \leftarrow DMMP(G_{t,j}, \mathbf{g}_{T\times N}, r, \alpha)$;
end for

{**Phase 2**: Search for the best DAG $\hat{S}$}
for every variable $G_{t,j} \in \mathbf{G}$ **do**
$\mathbf{P}_{[t-r,t],j} \leftarrow Best\_Subset(G_{t,j}, \mathbf{g}_{T\times N}, r, \mathbf{\Pi}_{[t-r,t],j}, \varphi, \gamma)$;
end for
return the highest scoring DAG $\hat{S}$ found;

our temporal variant of the MMPC algorithm for computing the maximum possible set of parents, $\mathbf{\Pi}_{[t-r,t],j}$, of a target variable $G_{t,j}$.

Algorithm 7.2 is the same as the MMPC algorithm of [196] except here the PC set is the set of maximum possible parents, $\mathbf{\Pi}_{[t-r,t],j} \subseteq \mathbf{G}_{[t-r,t]}$, of the target variable $G_{t,j}$. $\mathbf{G}_{[t-r,t]}$ is the set of all variables within the last r previous time-points $t-l$ for $1 \leq l \leq t$. Starting from an empty $\mathbf{\Pi}_{[t-r,t],j}$, Phase 1 of DMMP algorithm (7.2) sequentially adds variables $\Pi_{\lambda,\mu} \in \mathbf{G}_{[t-r,t]}$ which maximize the minimum association with the given target $G_{t,j}$ relative to the current $\mathbf{\Pi}_{[t-r,t],j}$. [196] proved that the $\mathbf{\Pi}_{[t-r,t],j}$ found in Phase 1 DMMP does not contain false negatives but may contain false positives which are then removed subsequently. As in [196], we define the minimum association between a variable $G_{l,i} \in \mathbf{G}_{[t-r,t]}$ and the target $G_{t,j}$ relative to a subset $\mathbf{Z} \subseteq \mathbf{G}_{[t-r,t]}$ as:

$$MinAssoc(G_{l,i}; G_{t,j}|\mathbf{Z}) = \min_{\mathbf{C}\subseteq\mathbf{Z}} Assoc(G_{l,i}; G_{t,j}|\mathbf{C}). \tag{7.39}$$

Algorithm 7.2 performs tests of independence $Ind(G_{l,i}; G_{t,j}|\mathbf{Z})$ which returns *true* if $G_{l,i}$ and $G_{t,j}$ are conditionally independent given $\mathbf{Z}$. The function $Assoc(G_{l,i}; G_{t,j}|\mathbf{Z})$ estimates the strength of dependency between $G_{l,i}$ and $G_{t,j}$ given $\mathbf{Z}$ such that $Assoc(G_{l,i}; G_{t,j}|\mathbf{Z}) \geq 0$ with equality holding if and only if $Ind(G_{l,i}; G_{t,j}|\mathbf{Z})$. For the independence tests $Ind(G_{l,i}; G_{t,j}|\mathbf{Z})$ we calculated the G^2 statistic as in [196] under the null hypothesis of the conditional inde-

Algorithm 7.2 The *DMMP* Algorithm

Input: $G_{t,j}$: target node
$\mathbf{g}_{T\times N} = (\mathbf{g}_1, \ldots, \mathbf{g}_T)^{\mathrm{T}}$: time-series data
r: maximum time delay
α: significance level
Output: $\mathbf{\Pi}_{[t-r,t],j}$: maximum possible parent set of $G_{t,j}$

{**Phase 1**: Compute a candidate $\mathbf{\Pi}_{[1,r],j}$}
$\mathbf{\Pi}_{[t-r,t],j} \leftarrow \emptyset$;
repeat
  $\varphi \leftarrow \max_{G_{l,i}\in\mathbf{G}_{[t-r,t]}} MinAssoc(G_{l,i}; G_{t,j}|\mathbf{\Pi}_{[t-r,t],j})$;
  $\Pi_{\lambda,\mu} \leftarrow \arg\max_{G_{l,i}\in\mathbf{G}_{[t-r,t]}} MinAssoc(G_{l,i}; G_{t,j}|\mathbf{\Pi}_{[1,r],j})$;
  if $\varphi \neq 0$ **then**
    $\mathbf{\Pi}_{[t-r,t],j} \leftarrow \mathbf{\Pi}_{[t-r,t],j} \cup \{\Pi_{\lambda,\mu}\}$;
  end if
until $\mathbf{\Pi}_{[t-r,t],j}$ has not changed

{**Phase 2**: Remove false positives}
for all $\Pi_{\lambda,\mu} \in \mathbf{\Pi}_{[t-r,t],j}$ **do**
  if $\exists \mathbf{\Gamma} \subseteq \mathbf{\Pi}_{[t-r,t],j}$ s.t. $Ind(\Pi_{\lambda,\mu}; G_{t,j}|\mathbf{\Gamma})$ **then**
    $\mathbf{\Pi}_{[t-r,t],j} \leftarrow \mathbf{\Pi}_{[t-r,t],j} \setminus \{\Pi_{\lambda,\mu}\}$;
  end if
end for
return $\mathbf{\Pi}_{[t-r,t],j}$;

pendence holding and using the same number of degrees of freedom as [196]. The G^2 returns a p-value and we reject the null hypothesis when p-value $< \alpha$. We set the significance level $\alpha = 0.05$ and define the measure of association as:

$$Assoc(G_{l,i}; G_{t,j}|\mathbf{Z}) = \begin{cases} 0 & \text{if } p\text{-value} \geq \alpha \\ \alpha - p\text{-value} & \text{otherwise} \end{cases}. \tag{7.40}$$

Since the gene expression time-series data are sparse and the number of counters goes exponentially as the number of parents increases, there probably are some zero cells in the contingency table when conducting conditional independence test. This may also lead the degree of freedom to be negative, which is a computational disaster when applying MMHC and its high-order extension to sparse gene time-series data. For example, we test $Ind(A, B|C)$ where nodes A, B and C have two states, respectively. The contingency table have 8 cells, and the degree of freedom is $1 \times 1 \times 2 = 2$. If there are more than 2 cells are

zeros, then the degree of freedom becomes negative. We propose a smooth method that is inspired by the computation of BDeu score [201]. We add a constant number, which is called *equivalent sample size* (ESS) or *pseudocount*, to each cell of the contingency table. In the current study, we uniformly set the ESS to 10, though ESS following other distributions is also possible.

After determining the set $\bar{\mathbf{\Pi}}_{j,[t-r,t]}$ of target variable $X_{j,t}$ $(1 \leq j \leq m)$, Phase 2 of MMHO-DBN will then perform a search-and-score strategy in order to find the best subset of parents $\mathbf{\Pi}_{j,[t-r,r]} \subseteq \bar{\mathbf{\Pi}}_{j,[t-r,t]}$ maximizing a score function (e.g., BDe, BDeu, BIC, etc). The search starts with an empty DAG and is constrained to consider only adding an edge "$\Pi_{\lambda,\mu} \rightarrow X_{j,t}$" if $\Pi_{\lambda,\mu} \in \bar{\mathbf{\Pi}}_{[j,t-r,t]}$; that is, the search for best subset $\mathbf{\Pi}_{j,[t-r,t]}$ is constrained to the set of possible parents $\bar{\mathbf{\Pi}}_{j,[t-r,t]}$ only. Our search algorithm is shown in Algorithm 7.3. In Algorithm 7.3, we used the BDeu or BIC metrics as the score function *Score*. The parameter γ is the maximum allowed cardinality of $\bar{\mathbf{\Pi}}_{j,[t-r,t]}$ below which we can perform an exhaustive search, otherwise we perform a heuristic search for best subset. In the algorithm, if the size of candidate parents are very large, we are only searching for subsets $\mathbf{\Pi}^{(\varphi)}$ with $|\mathbf{\Pi}^{(\varphi)}| \leq \varphi$ which maximize the score; essentially, parameter φ limits the size of the search space for the sake of computational efficiency. In our current implementation, we performed greedy search in this situation.

Algorithm 7.3 The *Best_Subset* Algorithm

Input: $G_{t,j}$: target node
$\mathbf{g}_{T\times N} = (\mathbf{g}_1, \ldots, \mathbf{g}_T)^{\mathrm{T}}$: time-series data
r: maximum time delay
$\mathbf{\Pi}_{[t-r,t],j}$: maximum possible parent set of $G_{t,j}$
φ: cardinality limit for $\mathbf{P}_{[t-r,t],j}$
γ: cardinality limit for exhaustive search
Output: $\mathbf{P}_{[t-r,t],j}$: best subset of parents of $G_{t,j}$

if $|\mathbf{\Pi}_{[t-r,t],j}| \leq \gamma$ **then**
{Exhaustive search}
$\mathbf{P}_{[t-r,t],j} \leftarrow \arg\max_{\mathbf{P}^{(\varphi)} \subseteq \mathbf{\Pi}_{[t-r,t],j}} Score(\mathbf{P}^{(\varphi)}, G_{t,j})$;
else
{Heuristic search}
$\mathbf{P}_{[t-r,t],j} \leftarrow Heuristic(G_{t,j}, \mathbf{\Pi}_{[t-r,t],j}, Score, \varphi)$;
end if
return $\mathbf{P}_{[t-r,t],j}$;

3.5 Preliminary Experiment

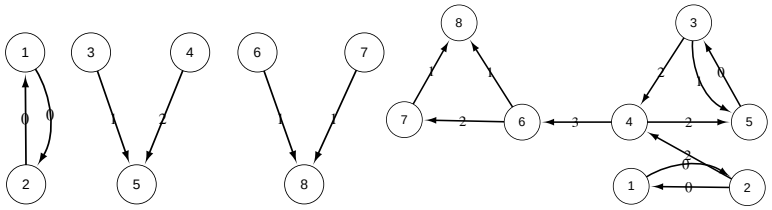
In this section, we investigate the performance of our MMHO-DBN approach in terms of accuracy and running time. We compared our MMHO-DBN, with DBmcmc (a first-order DBN) [218] and DBN-ZC [217] methods. We set the maximum-fan-in of each gene to 3 in all methods for fair comparison. We did two parts of experiments.

Our first experiment is to test whether ours can identify regulators of different time-delays. We designed a small network of 8 nodes as shown in Figure 7.5a. The time-delays are given along the directed connections. The expression values are discrete, and include 1 (down) and 2 (up). This network is composed of a pair of coherent nodes, regulators of different time-delays, and regulators of the same time-delay. Using this network, we generated a data set with 80 time points with equal sampling rate. In our MMHO-DBN, we set the maximum time-delay to 3. The significance level was $\alpha = 0.05$. The predicted networks by our MMHO-DBN, DBmcmc, and DBN-ZC are demonstrated in Figures 7.5b, 7.5c, and 7.5d, respectively. Their performances are compared in Table 7.3. The presence of a directed connection is defined as positive, and an absent edge is negative. From Figure 7.5, we can see that our MMHO-DBN can identify all the existing connections with correct time-delays, whereas DBmcmc can only predict the connections of 1 time-delay. DBN-ZC method fails to identify all existing connections, which convinces us that grouping regulators according to time-delays may not be a wise choice. DBN-ZC only searches among the subsets of the potential regulators with the *same* time-delays. As a high-order DBN, our approach runs very fast. In this experiment, it took only 11 seconds, while DBmcmc took 584 seconds.

Table 7.3: The comparison on simulated data.

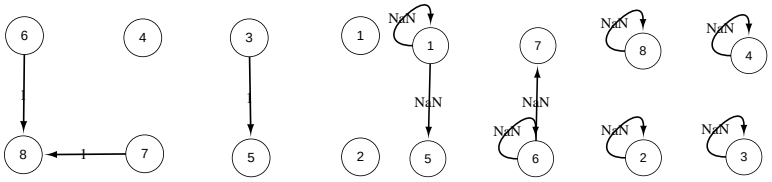
Method	Sensitivity	Specificity	Time (seconds)
MMHO-DBN	1	0.9138	11
DBmcmc	0.5	1	584
DBN-ZC	0	0.8621	0.4

In the second experiment, we ran our MMHO-DBN and the other methods on a yeast metabolic-cycle dataset [106]. We selected 44 genes that correspond to three periodical biological processes: Ox (oxidative), R/B (reductive, building), and R/C (reductive, charging). The data was sampled at 36 equally distributed time points. We set the maximum time-delays in MMHO-DBN to 1, 3, 5, and 7, respectively. The predicted GRNs are shown in Figure 7.6. We define the gene that regulates four or more genes as *hub gene*. For time-delays 1 and 3, genes POX1, FOX2, and MRPL10 are identified as hub genes by our



(a) Actual network.

(b) Learned by MMHO-DBN.



(c) Learned by DBmcmc.

(d) Learned by DBN-ZC. "NaN" means the time-lag information is unavailable in the implementation provided by the authors of [217].

Figure 7.5: The true network and the predicted networks.

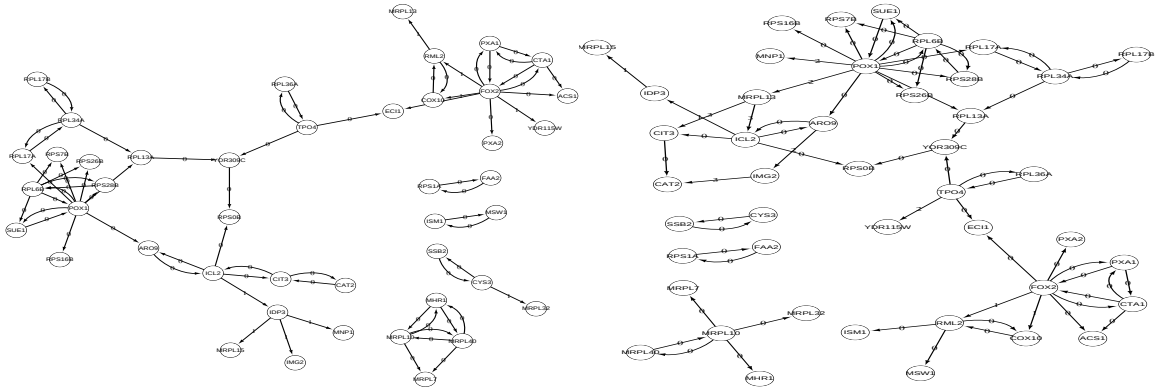
MMHO-DBN. For time-delay 5, SSB2 and RML2 are find as hub genes in addition to the three hub genes above using smaller time-delays. Using time-delay 7, we can predict CIT3, MRPL10, and RML2 to be hub genes. The GRN reconstructed by DBmcmc is shown in Figure 7.7. We can observe that CIT3, CAT2, ICL2, and RPSOB are predicted as hub genes using first-order method. The GRN identified by DBN-ZC is so sparse that only hubs POX1, MRPL10, FOX2, and few single regulators are found as can be seen in Figure 7.8. Comparing the results obtained by these methods, we can see that POX1, FOX2, MRPL10, and CIT3 are commonly recognized as hub genes. Since the actual GRN is usually unknown, there is no gold standard to validate the quality of predicted GRNs. Our current result is preliminary. We will find a reasonable validation method to further compare their performance. However, the advantage of our method can be felt in our first experiment on simulated data.

The running time of these methods are listed in Table 7.4. DBN-ZC is the fastest method, however as can be seen above the result of this method does not look better. Using time-lag 1, our MMHO-DBN is much faster than DBmcmc. As the maximum time-lag increase, the running time of our MMHO-DBN does not increase dramatically.

In this section, we propose a fast high-order dynamic Bayesian network learning method for reconstructing gene regulatory networks. This is a constraint-and-scoring method. In the algorithm, we also propose to use equivalent sample size to overcome the potential computa-

Table 7.4: The comparison of DBNs.

Method	Max Time-Lag	Time (seconds)
MMHO-DBN	1	147
	3	323
	5	478
	7	432
DBmcmc	1	379
DBN-ZC	-	71



(a) The maximum time-lag is 1 time unit.

(b) The maximum time-lag is 3 time unit.

(c) The maximum time-lag is 5 time unit.

(d) The maximum time-lag is 7 time unit.

Figure 7.6: The gene regulatory network learned by MMHO-DBN.

tional problem when testing the conditional independence. The preliminary experiment on simulated data shows that our method can identify regulators of different time-delays. The

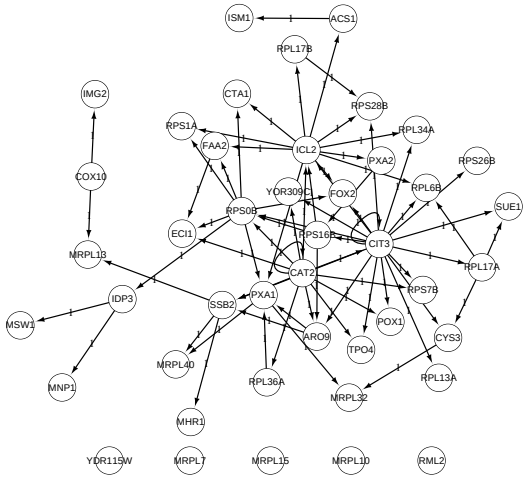


Figure 7.7: The gene regulatory network learned by DBmcmc.

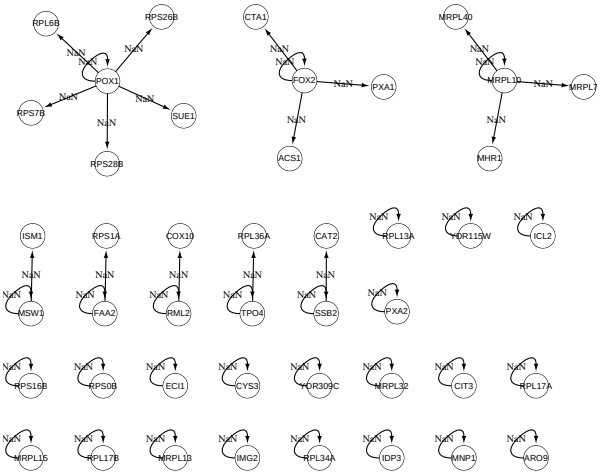


Figure 7.8: The gene regulatory network learned by DBN-ZC.

experimental result on real data proves that our approach is very efficient. We are currently working on a validation approach for comparing different network learning approaches on real gene expression time-series data.

4 Qualitative Probabilistic Networks

We start this section with quoting the definition of *qualitative probabilistic networks* (QPNs) given by Ibrahim, Ngom, and Tawfik [219]: *Qualitative probabilistic networks (QPNs) are DAGs that represent a qualitative abstraction of Bayesian networks. Formally, a QPN is*

given by a pair $G = (V(G), Q(G))$, where $V(G)$ is the set of nodes capturing random variables and $Q(G)$ is the set of arcs capturing the conditional dependence among the variables as in Bayesian Networks. Instead of a known conditional probability distribution however, the arcs of a QPN capture qualitative relations by finding monotonic characteristics in the local conditional probability distribution of each node based on the idea of first-order stochastic dominance. The resulting relations are used to establish properties over the probabilities of events and are of two types, binary qualitative influences and tertiary qualitative synergies.

Given the structure and parameter of a BN or DBN, we can identify the conditional dependency among variables. Sometimes, for example in GRN, pruning the network (that is removing the week and questionable connections) and qualitative interpretation may be necessary. We would like to construct the QPNs [220] from the learned BNs or DBNs to pursue this possibility. In this section, we first survey the basic concepts of qualitative influence and synergy. We then give the generalized concepts of synergy proposed in [219]. After that we survey the related works in the literature.

4.1 Qualitative Influence and Synergy

Qualitative *influences* describe the effects of the change of a variable to that of another variable [221, 220]. It includes positive, negative, neutral, and unknown influences. *Positive influence* of A on B means increasing the value of A makes a higher value of B more likely. It is given in the following definition:

Definition 1. *Given $A \in \mathbf{\Pi}(B)$ where $\mathbf{\Pi}(B)$ is the parent set of B, there is a positive influence from A on B, if and only if for $\forall a_1, a_2 \in val(A), a_2 > a_1$, for $\forall b_i \in val(B)$, and for $\forall \boldsymbol{x} \in val(\boldsymbol{X})$ where $\boldsymbol{X} = \mathbf{\Pi}(B) - \{A\}$, the following inequality*

$$p(B \geq b_i | a_2, \boldsymbol{x}) \geq p(B \geq b_i | a_1, \boldsymbol{x}) \tag{7.41}$$

is satisfied.

Negative influence means that increasing the value of A makes a lower value of B more likely. *Neutral influence* indicates that changing the value of A is does not affect the value of B. Negative and neutral influence are defined by changing "$\geq$" to "$\leq$" and "$=$", respectively. We can use $I^+(A, B)$ to denote that variable A has a *positive* influence on B. Similarly, negative influence and neutral are denoted by $I^-(A, B)$ and $I^0(A, B)$, respectively. Additionally, we can use $I^?(A, B)$ to denote that the influence of A on B is *unknown*.

Qualitative *synergies* define the joint influences of two variables on another variable. They include additive and product synergies. *Additive synergies* describe the joint influence of two parents on a common child. Additive synergies includes positive, negative, and neutral synergies. The *positive additive synergy* of two parents A and B on C implies that the joint influence of A and B is greater than the sum of the individual influences of A and B on C. It is defined as below:

Definition 2. *Given $A, B \in \mathbf{\Pi}(C)$, there is a positive additive synergy between A and B on C if and only if $\forall a_1, a_2 \in val(A), a_2 > a_1$, $\forall b_1, b_2 \in val(B), b_2 > b_1$, $\forall c_i \in val(C)$, and $\forall \boldsymbol{x} \in val(\boldsymbol{X})$ where $\boldsymbol{X} = \mathbf{\Pi}(C) - \{A, B\}$, the following inequality*

$$p(C \geq c_i|a_2, b_2, \boldsymbol{x}) + p(C \geq c_i|a_1, b_1, \boldsymbol{x}) \geq p(C \geq c_i|a_2, b_1, \boldsymbol{x}) + p(C \geq c_i|a_1, b_2, \boldsymbol{x}) \quad (7.42)$$

is satisfied.

The *negative additive synergy* of two parents A and B on child C implies that the joint influences of A and B is less than the sum of the individual influences of A and B on child C. *Neutral additive synergy* says that the joint influence is equal to the sum of individual influences. Negative and neutral additive synergy are defined by changing "$\geq$" to "$\leq$" and "$=$", respectively. We can use $S^+(\{A, B\}, C)$, $S^-(\{A, B\}, C)$, $S^0(\{A, B\}, C)$ to denote the positive, negative, and neutral additive synergies, respectively.

The additive synergies defined above works only for two parents. The authors of [219] have extended the positive additive synergy to the case of more than two parents. The *generalized positive additive synergy* is described by the following definition:

Definition 3. *Given $\boldsymbol{A} = [A_1, \cdots, A_q]$ and $\boldsymbol{A} \subseteq \mathbf{\Pi}(C)$, there is a positive additive synergy among $A_1, \cdots, A_q$ on C if and only if $\forall \boldsymbol{a}, \boldsymbol{a}' \in val(\boldsymbol{A})$ where $\boldsymbol{a} = [a_1, \cdots, a_q]$ and $\boldsymbol{a}' = [a'_1, \cdots, a'_q]$, $\boldsymbol{a} > \boldsymbol{a}'$, $\forall c_i \in val(C)$, and $\forall \boldsymbol{x} \in val(\boldsymbol{X})$ where $\boldsymbol{X} = \mathbf{\Pi}(C) - \boldsymbol{A}$, the following inequality*

$$p(C \geq c_i|\boldsymbol{a}, \boldsymbol{x}) + p(C \geq c_i|\boldsymbol{a}', \boldsymbol{x}) \geq \sum_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} p(C \geq c_i|\boldsymbol{\lambda}, \boldsymbol{x}) \quad (7.43)$$

is satisfied, where $\mathbf{\Lambda}$ is the set of all mixtures between $\boldsymbol{a}$ and $\boldsymbol{a}'$ excluding $\boldsymbol{a}$ and $\boldsymbol{a}'$. For example, if $q = 4$, $\boldsymbol{\lambda} = [a_1, a'_2, a'_3, a_4]$ is feasible.

Negative and neutral additive synergies can be generalized similarly. We use $S^+(\{A_1, \cdots, A_q\}, C)$, $S^-(\{A_1, \cdots, A_q\}, C)$, and $S^0(\{A_1, \cdots, A_q\}, C)$ to denote the generalized positive, negative, and neutral additive synergies, respectively.

4.2 Related Works

There are some efforts in literature to predict the signs of interactions. In [222], the signs of interactions and the corresponding influence magnitudes are computed based on the estimated CPTs. In [222], the idea of positive influence can be abstracted in the following definition:

Definition 4. *Given $A \in \mathbf{\Pi}(B)$, there is a positive influence from A on B if and only if for $\forall a_1, a_2 \in val(A), a_2 > a_1$, for $\forall b_i \in val(B)$, and for $\forall \boldsymbol{x} \in val(\boldsymbol{X})$ where $\boldsymbol{X} = \mathbf{\Pi}(B) - A$, the following inequality*

$$p(B \leq b_i | a_2, \boldsymbol{x}) \leq p(B \leq b_i | a_1, \boldsymbol{x}) \tag{7.44}$$

is satisfied.

In fact, this definition is equivalent to the standard definition of influence. This is because:

$$\begin{aligned}
&p(B \leq b_i | a_2, \boldsymbol{x}) \leq p(B \leq b_i | a_1, \boldsymbol{x}) \\
&\Leftrightarrow 1 - p(B > b_i | a_2, \boldsymbol{x}) \leq 1 - p(B > b_i | a_1, \boldsymbol{x}) \\
&\Leftrightarrow p(B > b_i | a_2, \boldsymbol{x}) \geq p(B > b_i | a_1, \boldsymbol{x}) \\
&\Leftrightarrow p(B > b_i | a_2, \boldsymbol{x}) + p(B = b_i | a_2, \boldsymbol{x}) \geq p(B > b_i | a_1, \boldsymbol{x}) + p(B = b_i | a_2, \boldsymbol{x}) \\
&\Leftrightarrow p(B \geq b_i | a_2, \boldsymbol{x}) \geq p(B \geq b_i | a_1, \boldsymbol{x}).
\end{aligned} \tag{7.45}$$

In [219], QPN is used to select candidate regulators before learning a high-order DBN. First, as in [217], candidate regulators of each variable is preliminarily obtained. Then, the set of regulators of each gene is refined by finding the maximum number of potential regulators which exhibit monotonic effects on the target genes. Each gene may have multiple sets of potential regulators corresponding to different generalized joint influences, respectively, and potential regulators within a subset may have different time-lags, and the time-lag information is associated to each subset.

4.3 Post-Analysis Using QPN

QPN can be applied to post-process the GRN reconstructed through DBN. We summarize the applications below:

1. The influences help us to predict the signs/types (repression and activation) of interactions.

2. The magnitude of the interactions can also be calculated.

3. Edges with zero or unknown influences can be pruned.

4. The additive synergy can help us to identify if a group of regulators work collaboratively or separately.

5. If we know the signs of interactions of a part of interactions, we can apply sign-propagation algorithms [221] to predict the signs of the remaining interactions.

Now, we give a method to compute the magnitude of influence. This method is similar to that in [222], but different in the definition of the cumulative distribution table and in computing the final magnitude. For a fixed value of the other parents $X = x_l$, suppose the CPT of A on B is denoted by matrix $\boldsymbol{\theta}$ of size $q \times r$, where q is the number of states of parent A, r is the number of states of child B, and $\theta_{ji} = p(B = val(B)_i | A = val(A)_j)$. Here we also suppose the states of each variable are ordered incrementally. The cumulative distribution table is denoted by $\boldsymbol{\Sigma}$, where $\sigma_{ji} = \sum_{k=i}^{r} \theta_{jk}$. The magnitude corresponding to this CPT is computed by

$$m_l = \frac{1}{r} \sum_{k=1}^{r} \sigma_{qk} - \sigma_{1k}, \tag{7.46}$$

which is the average difference between the last row and the first row of the cumulative distribution table. Then, the magnitude of influence of A on B is defined as

$$m = \frac{1}{L} \sum_{l=1}^{L} m_l, \tag{7.47}$$

where L is the number of states of other parents X.

5 Implementation

We implemented our MMHO-DBN method in MATLAB, and have assembled them into our *probabilistic graphical models* (PGM) Toolbox: `https://sites.google.com/site/pgmtool`. The current implementations mainly include BDeu and BIC scores, the two phases of MMHO-DBN, influence, and synergy. The list of these functions is given in Table 7.5.

Table 7.5: Current implementations of our probabilistic graphical models toolbox.

Function	Description	Example
computeBDeu	Compute the BDeu score of a node conditioned by its parents.	see the code
computeBIC	Compute the BIC score of a node conditioned by its parents.	see the code
computeScore	Compute the BDeu or BIC score of a network.	exampleComputeScore
searchCPAll	Phase I of MMHO-DBN: Search candidate parents of each node.	mainMMHODBN
searchParent	Phase II of MMHO-DBN: Search the parents of each node by a heuristic.	mainMMHODBN
computePerformance	Compute the accuracy etc. of the learned network.	mainMMHODBN
computeInfluence	Compute influences of connections given a learned network.	mainMMHODBN
computeSynergy	Compute synergies of parents given a learned network.	mainMMHODBN
influenceLocal	Compute the influences of the parents on a child.	-
synergyLocal	Compute the additive ynergies of the parents on a child.	-

6 New Computational Experiments [3]

In Section 3.5, we did preliminary experiments on synthetic and real-life data. On the real-life data, we did not validate the performance of MMHO-DBN, because we did not know the actual GRN. In this section, we validate the performance of our MMHO-DBN approach with QPN post-analysis on new synthetic and real-life data, whose actual networks are known. We compared our MMHO-DBN, with DBmcmc (a first-order DBN) [218] and DBN-ZC [217] methods. We set the maximum fan-in of each gene to 3 in all methods for fair comparison. We did two parts of experiments on simulated data and real-life gene expression time-series data, respectively.

We use sensitivity, precision, and F-measure to evaluate the performance of the methods. The presence of a directed connection is defined as positive, and an absent edge is negative. We denote the numbers of true positives, true negatives, false positives, and false negatives as TP, TN, FP, and FN, respectively. The sensitivity (also called *recall*) is defined as $\frac{TP}{TP+FN}$. The precision (also called *positive predictive value*) is defined as $\frac{TP}{TP+FP}$. The F-measure is defined as $2 * \frac{presion*sensitivity}{precision+sensitivity}$. When evaluating the performance of the influence in the post-analysis after obtaining a DBN, we define the accuracy of influence as

[3]The experiment is our joint research with Dr. Jie Zheng and Haifen Chen of National Technological University, Singapore. This work will be submitted to the Journal of Computational Biology.

$\frac{\sum_{i=1}^{TP} s_i}{TP}$, where s_i is defined as

$$s_i = \begin{cases} 1 & \text{if } I_p(e_i) = I_a(e_i) \\ 0.5 & \text{if } (I_p(e_i) \neq I_a(e_i)) \text{ and } (I_a(e_i) =? \text{ or } I_p(e_i) =?) \\ 0 & \text{otherwise} \end{cases}, \tag{7.48}$$

Here, $I_a(e_i)$ returns +, −, 0, or ? for positive, negative, neutral, and unknown influences of the true positive edge e_i on the actual network. $I_p(e_i)$ is defined analogously on the predicted network.

6.1 On Simulated Data

Our first experiment has two purposes. First, we want to know whether MMHO-DBN can identify regulators of different time-delays. Second, we test if different influences and synergies can be identified correctly. We designed a network of 8 nodes as shown in Figure 7.9. This network is composed of regulators of different time-delays and regulators of the same time-delay. The time delays, types of influences, and magnitudes of influences are given along the directed connections in the format of "time lag, type and magnitude of influence". The parameters, that is, the (conditional) probability tables are also given in this figure. The values of the network are discrete, and include 1 (down) and 2 (up). From the parameters, we can see that there are two additive synergies: $S^+(\{X_1, X_2\}, X_3)$ and $S^+(\{X_3, X_6\}, X_7)$.

Using this network, we generated a data set of 200 time points with equal sampling rate. In our MMHO-DBN, we set the maximum time-delay to 2. The significance level was $\alpha = 0.05$ in the conditional independence test. We used BDeu metric to score a structure. The equivalent sample size is set to 1 for both conditional independence test and BDeu score. The predicted networks by MMHO-DBN, DBmcmc, and DBN-ZC are demonstrated in Figures 7.10a, 7.10b, and 7.10c, respectively. Their performances including precisions, sensitivities, and F-measures, are compared in Table 7.6.

From Figure 7.10, first of all, we can see that our MMHO-DBN can identify all the existing connections with correct time-delays, whereas DBmcmc can only predict the connections of 1 time-delay correctly. DBN-ZC method fails to identify all existing connections, which convinces us that grouping regulators according to time-delays may not be a wise choice in some cases. DBN-ZC only searches among the subsets of the potential regulators with the *same* time-delays. Moreover, the time-lag estimation through measuring the initial changes may not make more sense on noisy and truncated data. Second, the post-processing

of MMHO-DBN can predict all types of influences correctly. The magnitudes of influences are also very close to the true strengthes which are computed by using the true CPTs. Furthermore, the two true additive synergies are predicted correctly by the post-analysis of MMHO-DBN. As a high-order DBN, our approach runs very fast. It can be seen in Table 7.6 that, MMHO-DBN took only 34 seconds, while DBmcmc took 1878 seconds.

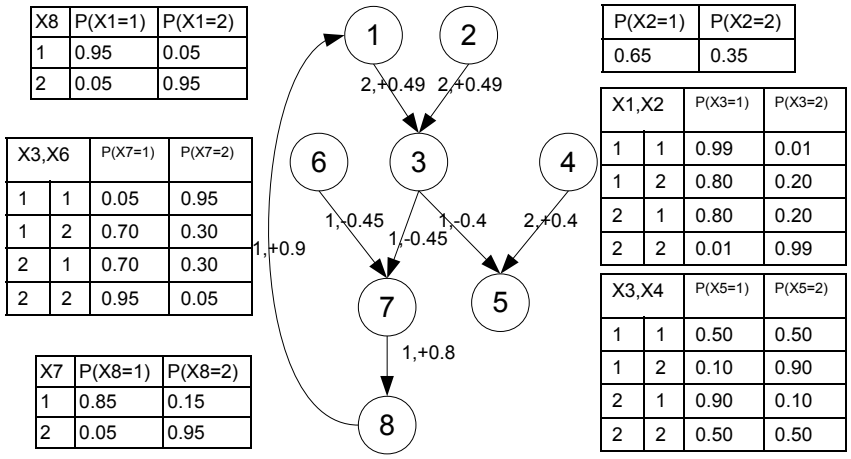


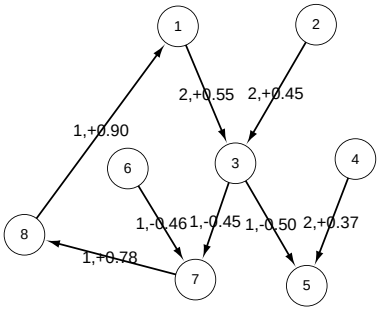
Figure 7.9: The true network from which simulated data are sampled.

Table 7.6: Comparison on simulated data.

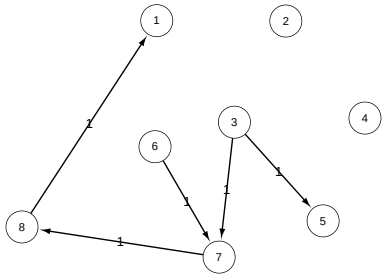
Method	Precision	Sensitivity	F-measure	Time (seconds)
MMHO-DBN	1	1	1	34
DBmcmc	1	0.6250	0.7692	1878
DBN-ZC	0	0	-	0.355

Sensitivity Analysis

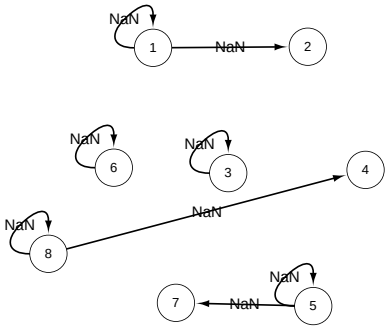
In the MMHO-DBN model, there are a few parameters. The first parameter is the equivalent sample size, which is used in the calculations of conditional independence test and BDeu score, respectively. A larger equivalent sample size would make the corresponding distribution smoother. The second parameter is the maximum time-delay (that is r in Algorithm 7.1), which specifies the largest time delay of the regulations considered when inferring the regulatory relations among genes. On one hand, the larger the maximum time-delay is, the higher the order of MMHO-DBN would be, and gene regulations with longer range could be detected. On another hand, a larger maximum time-delay will result in a smaller sample size. The third parameter is the maximum number of parents for each



(a) Predicted by MMHO-DBN.



(b) Predicted by DBmcmc.



(c) Predicted by DBN-ZC. "NaN" means the time-lag information is unavailable in the implementation provided by the authors of [217].

Figure 7.10: The predicted networks using simulated data.

gene (that is φ in Algorithm 7.3), i.e. the maximum fan-in of each node. A larger maximum number of parents would allow more multiple regulations being discovered, although it may require more running time, and the number of the corresponding parameters would increase exponentially. Generally, we set the equivalent sample sizes to 1, maximum time-delay to 2, and maximum parents to 3. Here, we investigate how the parameters affect the performance of MMHO-DBN with respect to the F-measure. This can help us set the parameters properly for future applications.

We first investigated how the parameter equivalent sample size affects the performance of MMHO-DBN. The equivalent sample size is used, respectively, in the phases of obtaining candidate parents and searching the parents for each node. For the convenience of narration, we denote *equivSize1* and *equivSize2* as the equivalent sample sizes in these processes, respectively. We kept the maximum time-delay and the maximum number of parents fixed to 2 and 3, respectively. MMHO-DBN was run with *equivSize1* and *equivSize2* ranging from

1 to 15, respectively. The variation of the performance of MMHO-DBN is shown in the top-left sub-plot of Figure 7.11. We can see that, as *equivSize1* increases, the performance of MMHO-DBN does not change. This implies that the performance of the algorithm is insensitive to the value of *equivSize1* on the simulated data. However, a larger *equivSize2* is likely to decrease the performance of MMHO-DBN.

Second, in order to examine how the maximum time-delay affects the performance of MMHO-DBN, we ran MMHO-DBN with the maximum time-delay from 1 to 100. Meanwhile, we fixed both *equivSize1* and *equivSize2* to 1, and the maximum number of parents to 3. The result is shown at the bottom of Figure 7.11. We can see that, when the maximum time-delay is 1, the algorithm did not reach its best performance. The best performance was obtained when the maximum time-delay was greater than 1. As the maximum time-delay increases, the performance gets worse. Simply put, the maximum time-delay corresponding to the best performance is consistent with the actual maximum time-delays, as compared in Figure 7.9.

Similarly, in order to study the effect of the maximum number of parents on the performance of MMHO-DBN, We ran MMHO-DBN with the maximum number of parents changing from 1 to 5. Meanwhile, we fixed both *equivSize1* and *equivSize2* to 1, and the maximum time-delay to 2. The result is shown in the top-right sub-figure of Figure 7.11. We can find that the algorithm is insensitive to this parameter on the simulated data.

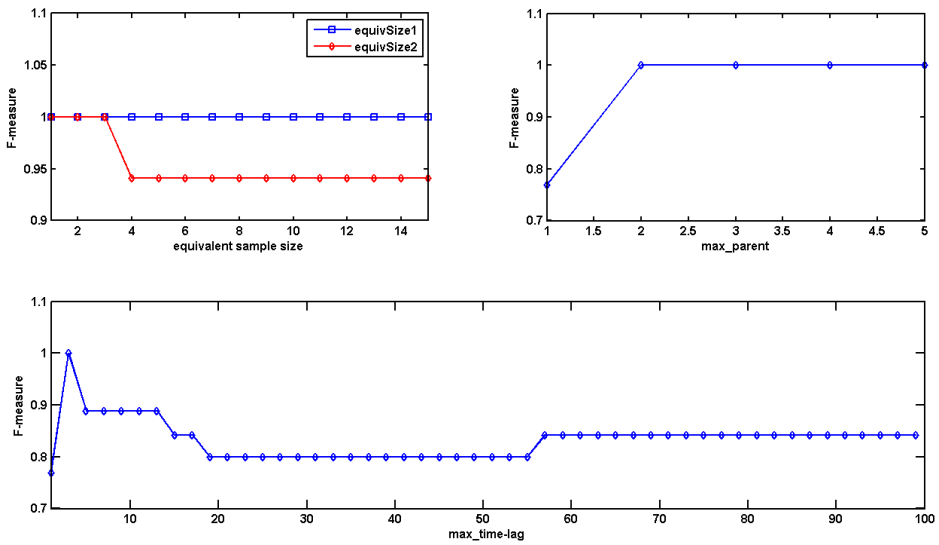
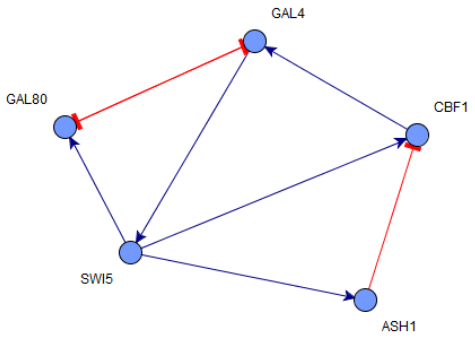


Figure 7.11: Effect of parameters on the performance of MMHO-DBN on the simulated data.

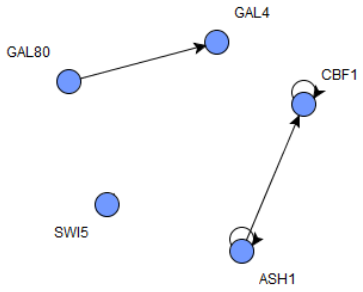
6.2 On Real-Life Data

We have validated our method on two real-life benchmark networks of yeast. The first benchmark network has five genes, which is a synthetic but real-life network for *in vivo* reverse-engineering and modeling assessment (IRMA) [223]. This is one of the first attempts to build a real-life gene regulatory network with accurately known gene interactions and reference data of gene expression. In their significant work, the authors constructed a network of five genes of yeast *Saccharomyces cerevisiae*. The regulatory interactions among these five genes were carefully designed. The network can be triggered by galactose. This network would be "switched" on when the cell culture is shifted from glucose to galactose, and "switched" off if the culture is shifted from galactose to glucose. Two sets of time-series gene expression data were measured after the cell culture is perturbed: one named switch-on data set with 16 equally distributed time points (or *Yeast5on* for short), sampled after shifting the culture from glucose to galactose; and another named switch-off data set with 21 equally distributed time points (or *Yeast5off* for short), sampled after shifting the culture from galactose to glucose. The second benchmark network is a nine-gene network of yeast related to yeast cell cycle, identified by Simon *et al.* [224]. Each of the nine genes is a cell-cycle transcription activator, and this network shows how the transcription activators which function at one stage of the yeast cell cycle, regulate the transcription activators, which are active at the next stage. The gene expression data of yeast cell cycle are obtained from [225], which is a time-series data set sampled at 24 equally distributed time points. This data set is abbreviated as *Yeast9* in our study. All these data were discretized to three states. We also tried two states, where the performance is very similar to that of using three states. Thus, we only report the performance of using three states below.

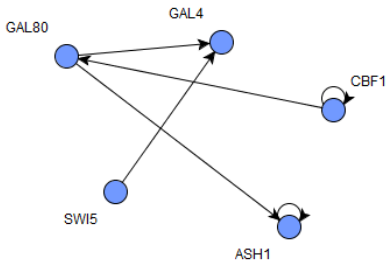
We have investigated the performance of our MMHO-DBN method on these data sets. To show the ability of MMHO-DBN in learning regulations with delayed interactions, we here compare the performance of MMHO-DBN, DBmcmc (a first-order DBN), and DBN-ZC (a high-order DBN). Table 7.7 shows their comparisons on the three real-life data sets, respectively. First, as seen, MMHO-DBN has a better performance than DBmcmc on all real-life data sets, which is probably because MMHO-DBN considers simultaneously gene regulations with longer time-delays, while DBmcmc only takes into account genetic interactions with one time delay. Second, MMHO-DBN also performed better than DBN-ZC. We also tested the first-order MMHO-DBN (by setting the parameter maximum time-delay to 1) on the three real data sets and compared with DBmcmc. The results are shown in Table 7.8. We can see that even the first-order MMHO-DBN has competitive performance compared to DBmcmc.



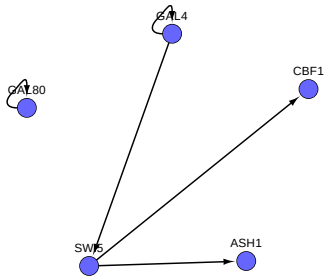
(a) Actual *Yeast5* network.



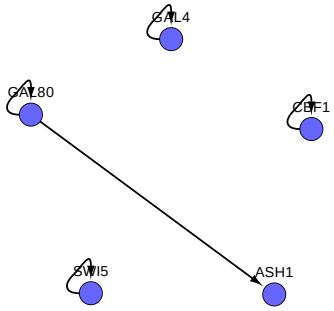
(b) DBmcmc on *Yeast5on*.



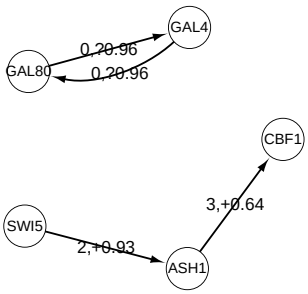
(c) DBmcmc on *Yeast5off*.



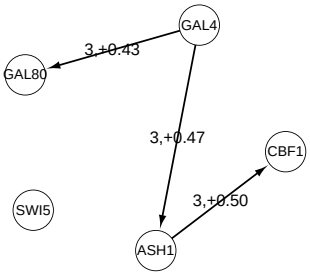
(d) DBN-ZC on *Yeast5on*.



(e) DBN-ZC on *Yeast5off*.

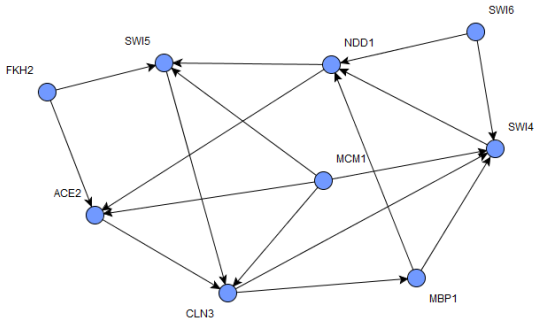


(f) MMHO-DBN on *Yeast5on*.

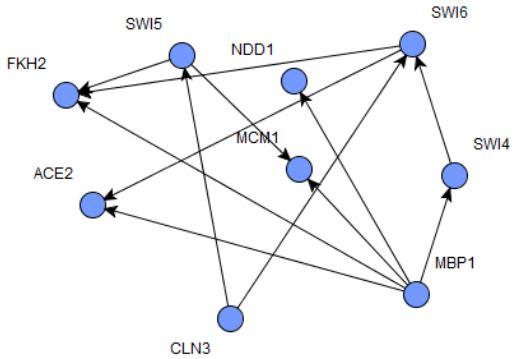


(g) MMHO-DBN on *Yeast5off*.

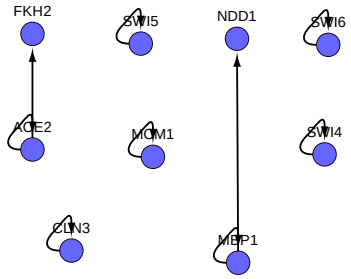
Figure 7.12: The actual *Yeast5* networks and the predicted networks using *Yeast5* data.



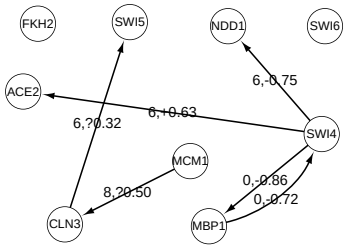
(a) Actual *Yeast9* network.



(b) DBmcmc on *Yeast9*.



(c) DBN-ZC on *Yeast9*.



(d) MMHO-DBN on *Yeast9*.

Figure 7.13: The actual *Yeast9* network and the predicted networks using *Yeast9* data.

Table 7.7: Comparison on real-life data.

Data	Method	Precision	Sensitivity	F-measure	Time (seconds)
Yeast5on	MMHO-DBN	1.0000	0.5000	0.6667	1
	DBmcmc	0.5000	0.2500	0.3333	108
	DBN-ZC	0.6000	0.3750	0.4615	0.03
Yeast5off	MMHO-DBN	0.6667	0.2500	0.3636	2
	DBmcmc	0.1700	0.1200	0.1407	109
	DBN-ZC	0	0	NA	0.1
Yeast9	MMHO-DBN	0.5000	0.1765	0.2609	28
	DBmcmc	0.2100	0.1400	0.1680	140
	DBN-ZC	0.1111	0.0588	0.0769	0.2

Table 7.8: Comparison of MMHO-DBN (first-order) and DBNmcmc on real-life data.

Data	Method	Precision	Sensitivity	F-measure
Yeast5on	MMHO-DBN	1.0000	0.2500	0.4000
	DBmcmc	0.5000	0.2500	0.3333
Yeast5off	MMHO-DBN	0.4000	0.2500	0.3077
	DBmcmc	0.1700	0.1200	0.1407
Yeast9	MMHO-DBN	0.2857	0.1176	0.1667
	DBmcmc	0.2100	0.1400	0.1680

The actual networks and the predicted networks by different methods are illustrated in Figures 7.12 and 7.13. Comparing the networks predicted by MMHO-DBN and the actual network of *Yeast5*, we find that the influence accuracy (defined in Equation (7.48)) of the QPN is 0.5. Unfortunately, the types of interactions on *Yeast9* is unavailable, so we can not validate the influence on this data. Moreover, we have no information to validate the synergies so far.

According to [224], genes SWI4 and MBP1 are both active in the phase "late G1" of the cell cycle; NDD1 is active in the phase "G2", and MCM1 is active in the phase "G2" or "M"; and SWI5 is active in the phase "M"; CLN3 is active in the phase "G1". Such information can serve as evidence for the delayed regulations between those genes. We have compared the time delays predicted by MMHO-DBN with those in real cases. The results are shown in Table 7.9. Note that the predicted time delays are measured in number of time lags. As seen, the predicted time delays are consistent with real cases.

Table 7.9: Validating the time-delays on *Yeast9*.

True Positive Regulation	Phases	Predicted Time Lags	Remark
SWI4→MBP1	Late G1→late G1	0	
SWI4→NDD1	Late G1→G2	6	
MCM1→CLN3	G2/M→G1	8	
CLN3→SWI5	G1→M	6	opposite direction

Sensitivity Analysis

We have also studied how the parameters affect the performance of MMHO-DBN on real-life data sets. We observed how MMHO-DBN behaved with equivalent sample size changing from 1 to 15 on data set *Yeast5on*. Meanwhile, we fixed the maximum time-delay and the maximum number of parents to 2 and 3, respectively. Also, the performance of MMHO-DBN was investigated with the maximum time-delay changing from 1 to 15, while the equivalent sample sizes and the maximum number of parents are fixed to 1 and 3, respectively.

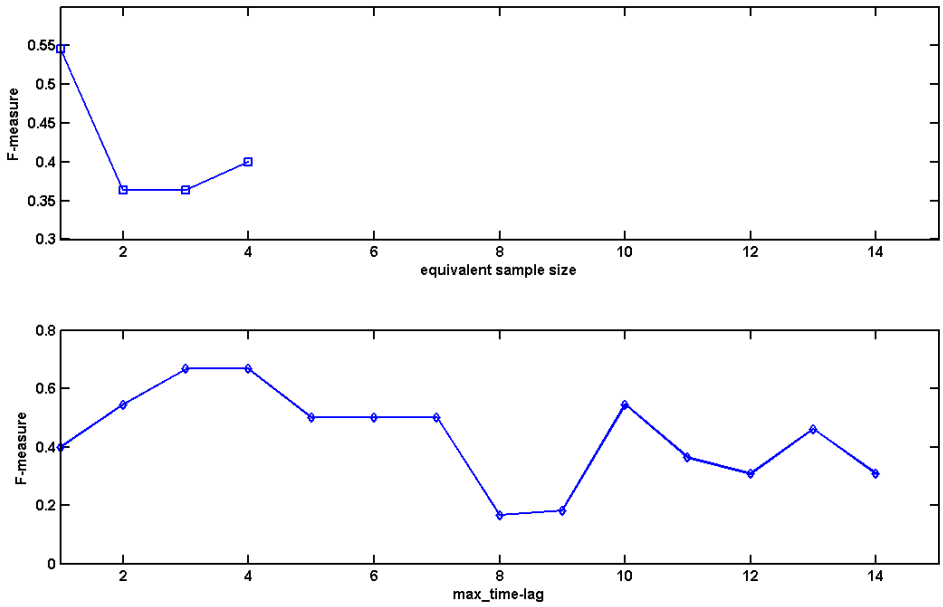


Figure 7.14: Effect of parameters on the performance of MMHO-DBN when applied on *Yeast5on* data.

Figures 7.14, 7.15, and 7.16 show the performance variations of MMHO-DBN with different parameters on *Yeast5on*, *Yeast5off*, and *Yeast9*, respectively. The missing data point in some of the sub-figures is because the corresponding F-measure is invalid (because

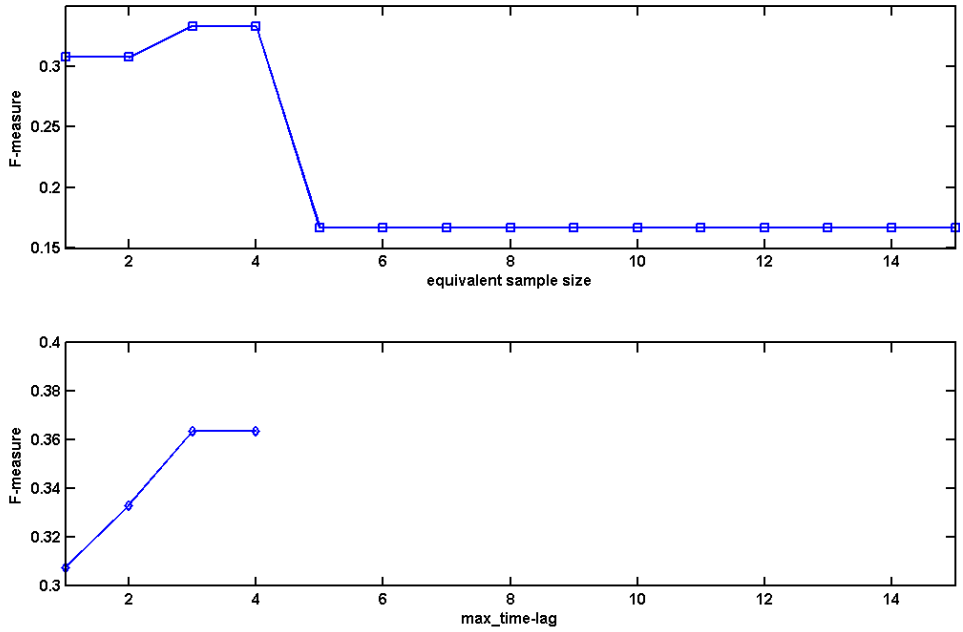


Figure 7.15: Effect of parameters on the performance of MMHO-DBN when applied on *Yeast5off* data.

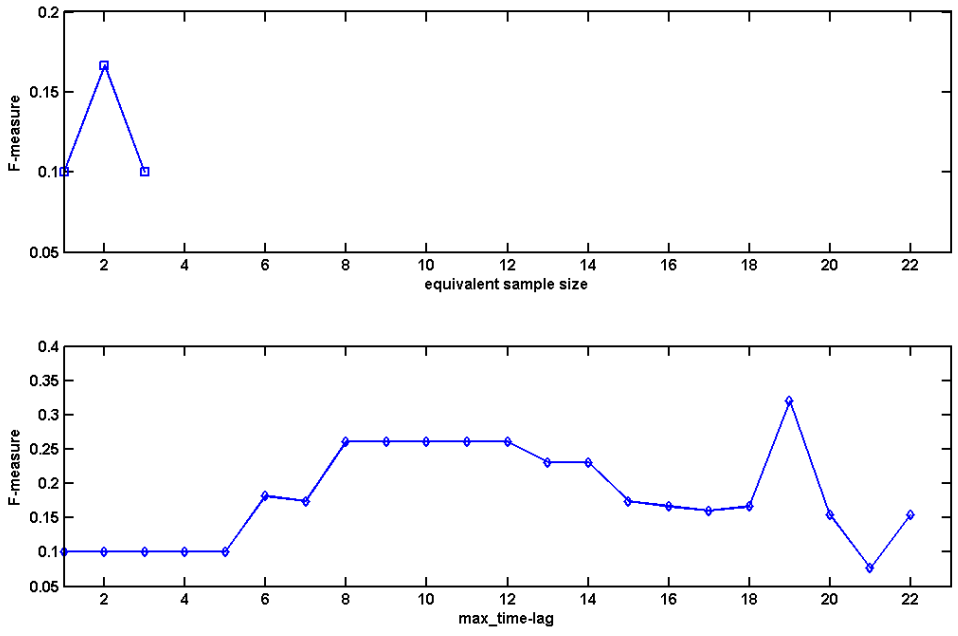


Figure 7.16: Effect of parameters on the performance of MMHO-DBN when applied on *Yeast9* data.

$TP = 0$). The effect of the maximum number of parents is not shown here, because we observed that it does not affect the performance of MMHO-DBN on the real-life data sets. For the equivalent sample size, we can see that good performance is obtained at the very beginning of the plot. This is because the sample size of gene time-series data is relatively small, and a large equivalent sample size would make the contingency table (in the conditional independence test) and the frequency table (in the BDeu score) very smooth. Thus the performance of MMHO-DBN gets worse. From the plots of the maximum time-delay, we observe that the best performance is not obtained with a few, rather than 1, time lags. The performance becomes unstable when the maximum time-delay is very large. The reason is that a very small maximum time-delay would prevent the algorithm from finding high-order interactions, while a very large value would further reduce the number of available samples to learning a reliable model.

From Figures 7.14, 7.15, and 7.15, we could draw the following conclusions about the parameter setting: (1) For equivalent sample size, a small value (≤ 3) may be appropriate; (2) The setting of maximum time-delay depends on the data, which means we would better use some prior knowledge about the data for an appropriate value of maximum time-delay; (3) Although here the the maximum number of parents does not affect MMHO-DBN on real data sets, a small value (≤ 3) may be a proper choice for the setting of maximum parents, considering the the number of time points and the complexity of the network. However, we are also interested in the challenge, as future work, that a true network has some nodes which have many ($\gg 3$) parents, while the number of time points is limited.

7 Conclusions

Learning a graphical model from microarray time-series data is one of the methods to infer gene regulatory networks. In this chapter, we proposed the MMHO-DBN method to reconstruct time-delayed regulations. We also applied the concepts of influence and synergy in QPN to infer the types of interactions and the joint effort of regulators on a target. Our experimental results on both synthetic data and real-life time-series data revealed that our MMHO-DBN method is able to identify more interactions precisely than other methods. We also conducted sensitivity analysis on the parameters of the MMHO-DBN. Based on this, suggestions have been given to choose the parameters properly.

There are many future works in this research. For example, we need to find a way to validate the synergies among regulators biologically. More actual regulatory networks are required to evaluate the performance of our method. Low accuracy is a still common

drawback of most of the models on time-series data, therefore we are considering combine knowledge from other data, for example the next generation sequence data.

Publications

1. We preliminarily proposed the MMHO-DBN learning algorithm for reconstruct time-delayed gene regulatory networks in [*Y. Li and A. Ngom, "The max-min high-order dynamic Bayesian network learning for identifying gene regulatory networks from time-series microarray data," IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB/SSCI), Singapore, Apr. 2013, pp. 83-90.*].

2. The comprehensive validation of the MMHO-DBN method was a joint research with Dr. J. Zheng and H. Chen of National Technological University, Singapore. This work will be submitted to the Journal of Computational Biology.

3. We implemented the MMHO-DBN and QPNs in MATLAB, which is available online [`https://sites.google.com/site/pgmtool`].

Chapter 8

Dealing With Missing Values

1 Introduction

Microarray data often suffer from *missing values* due to many factors such as insufficient resolution, image corruption, artifacts, systematic errors, or incomplete experiments. Microarray data analysis methods based on machine learning (such as clustering, dimension reduction, and classification) often require complete gene expression data sets, in order to perform robustly and effectively. Therefore, incomplete data sets need to be pre-processed for these methods before analysis, or handled carefully during their process. *Gene-sample-time* (GST) microarray data sets suffer from even more severe missing value problems. For example, if a patient is not available to obtain a sample at a certain time point, a whole vector for this patient is thus missing in the data set.

Contributions: In this chapter, we explore various strategies for learning models from incomplete biological data. Our main contributions include:

1. We extend the existing imputation methods, originally for two-way data, to methods for GST data.

2. We also propose a pair-wise weighting method for computing kernel matrices from incomplete data.

In the rest of this chapter, we first survey strategies of dealing with the missing value problems for two-way data and GST data in Sections 2 and secMissThree, respectively. After that, we propose a pair-wise weighting strategy to compute kernel matrix from vectorial samples with missing values in Section 4. We then extend the KNNimpute and SVDimpute,

originally devised for two-way data, for estimating missing values of GST data in Section 5.

2 Dealing With Missing Values in Two-Way Data [1]

There are *three* strategies for handling the issue of missing values in two-way data. First, we can remove the features or samples (time points) with missing values. However, the main drawback is that the already small sample (time point) size becomes smaller and we may face the risk of having not enough data for learning a model. The second strategy is to impute (that is, to estimate then fill-in) the missing values in the data [108]. Missing values can be either imputed by a constant value (e.g., 0) or by feature averages, or alternatively they can be estimated by some statistical or machine learning methods. This strategy essentially completes an incomplete data set, and hence, avoids deleting features or samples (or time points). The missing values can be estimated either before any analysis or during the learning of a model. The time complexity for making the data complete before learning is usually much lower than that of combining missing value estimation with learning a model, for example using *expectation maximization* (EM). The third strategy is that the observed values are only used during the learning and prediction processes. Thus, it is usually called weighting strategy. For incomplete training data $\boldsymbol{D}$, the objective of a weighted regression method for fitting a model can be expressed as

$$\min_{\boldsymbol{M}} \frac{1}{2}\|\boldsymbol{W} * (\boldsymbol{D} - \boldsymbol{M})\|_F^2, \tag{8.1}$$

where $\boldsymbol{M}$ contains the values estimated by the model, $\boldsymbol{W}$ is a weighting matrix with $w_{ij} = 1$ indicating that value d_{ij} is present in the data and $w_{ij} = 0$ indicating that d_{ij} is missing. Notation "$*$" is the Hadamard (element-wise) product operator. $\|\boldsymbol{A}\|_F$ is defined as the Frobenius norm of matrix $\boldsymbol{A}$.

Some imputation methods have been proposed specifically for two-way microarray data sets since the work of [108] in 2001. An imputation method that estimates a missing value by feature average value was investigated in [108], and was shown to give the worst performance (with respect to the normalized root mean squared error) among many imputation methods. In [108], a k-nearest-neighbor based imputation method, or *KNNimpute* for short, has been proposed using the Euclidean, Pearson correlation, and variance minimization similarity metrics, and the Euclidean metric gave the best results. For each incomplete gene g_i

[1] This section is based on our publication [111].

with missing value g_{ij} in the j-th sample or time point, imputation is done by first finding its k nearest complete genes, and then taking the weighted average value at column j of those k genes as estimation of g_{ij}. The authors of [108] also describes a singular-value-decomposition-based imputation method, *SVDimpute*. A set of eigen-genes is found by applying SVD on the complete genes only. Each incomplete gene is then represented as a linear combination of those eigen-genes. Linear regression using the *expectation maximization* (EM) algorithm is performed on those eigen-genes to estimate the missing values of given incomplete genes. *LLSimpute*, a local least squares method [226], represents an incomplete gene as a linear combination of its k nearest complete neighbors. Least squares optimization is used to find the coefficients of the linear combination, which are then used for estimating the missing values of the incomplete gene. Other methods based on least squares regression are also introduced in [227, 228, 229]. The Bayesian principal component analysis method, *BPCAimpute* [230], applies PCA similarly to the SVD method. However, an EM-like Bayesian estimation algorithm is used to estimate the coefficients of the linear combinations for each incomplete gene to impute. In [231], genes are represented as cubic spline functions first, and then missing values for incomplete genes are estimated by resampling the continuous curves. An autoregressive-model-based missing value estimation method (*ARLSimpute*) [232] first, applies auto-regression on a set of k similar genes (missing values are set to zeros initially) to estimate their AR coefficients by means of a least squares method. Then, using the AR coefficients, missing values for all incomplete genes are imputed by means of another least squares error regression method. ARLSimpute is the only method devised specifically for time-series profiles; however it works only on long stationary time-series data. It is also the only method that is able to impute a time point (entirely missing column of a gene-time microarray data). Except the approach in [231], imputation methods for two-way microarray data are all based on similar principles: they either find nearest complete genes to impute incomplete genes, or find eigen-genes to impute incomplete genes, or combine these two principles. Current methods, except the ARLSimpute method, are initially devised for static data, though they have been applied to time-series data. They neither work when an entire column of a two-way microarray data is missing.

Weighted non-negative matrix factorization (WNMF) is an implementation of the weighting strategy [96, 99]. Suppose the non-negative incomplete data $\boldsymbol{D}$ is to be decomposed into two non-negative matrices as $\boldsymbol{D} \approx \boldsymbol{AY}$. The weighted multiplicative update rules of

WNMF are

$$\begin{cases} \boldsymbol{A} = \boldsymbol{A} * \frac{(\boldsymbol{W} * \boldsymbol{D})\boldsymbol{Y}^{\mathrm{T}}}{(\boldsymbol{W} * \boldsymbol{A}\boldsymbol{Y})\boldsymbol{Y}^{\mathrm{T}}} \\ \boldsymbol{Y} = \boldsymbol{Y} * \frac{\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{W} * \boldsymbol{D})}{\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{W} * \boldsymbol{A}\boldsymbol{Y})} \end{cases}, \tag{8.2}$$

where $\boldsymbol{W}$ is a weighting matrix, as defined in Equation 8.1.

3 Dealing With Missing Values in GST Data [2]

We recommend the readers refer to the appendix of Chapter 4 for a short introduction to tensor algebra with useful illustrations. It is even more important to handle missing values for GST data by imputation or weighting, because if the removal strategy is used and a value is missing in the three-way data set, the corresponding gene-slice, sample-slice, or time-slice should be deleted for completeness. If there are many missing values distributed randomly in a GST data set, by removing a slice for each missing value, there may not be sufficient complete data left for analysis. Furthermore, many data mining approaches require complete data.

Similar to the EM-based SVD imputation method for two-way data, tensor factorization can be used to impute missing values for GST data. An iterative imputation method using PARAFAC decomposition is given in [233, 234]. The basic idea is the following. The missing values in the tensor data $\boldsymbol{\mathcal{D}}$ are first initialized by random numbers. Next, $\boldsymbol{\mathcal{D}}$ is factorized into $\boldsymbol{\mathcal{D}} = [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!]$ by PARAFAC. After that, the missing values are replaced with $d_{ijk} = \sum_{f=1}^{R} a_{if} b_{jf} c_{kf}$, where R is the tensor rank. The above steps are repeated until there is no change in the estimated values. The PARAFAC-alternating least squares with single imputation (PARAFAC-ALS-SI) [235] is an implementation of the above idea using the ALS algorithm. If $\boldsymbol{\mathcal{D}}$ is non-negative, it is necessary that the factors $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ must be restricted by non-negativity. In this situation, non-negative PARAFAC should be used [236, 237, 234].

The general weighted regression model for a three-way data set is as follows:

$$\min_{\boldsymbol{\mathcal{M}}} \frac{1}{2} \|\boldsymbol{\mathcal{W}} * (\boldsymbol{\mathcal{D}} - \boldsymbol{\mathcal{M}})\|_F^2. \tag{8.3}$$

The model to be learned may be constrained by non-negativity, orthogonality, or others. The *incomplete data PARAFAC* algorithm (INDAFAC) [235] is an implementation to fit the PARAFAC model using a weighting strategy. Sparse non-negative Tucker algorithm

[2] This section is based on our publication [111].

(SN-TUCKER or HONMF) [123] is the extension of weighted sparse *non-negative matrix factorization* (NMF) into non-negative Tucker decomposition that can ignore missing values.

4 Pair-Wise Weighting Method [3]

4.1 Pair-Wise Weighting is A Local Method

The kernel sparse coding, kernel dictionary learning, and kernel linear models, described in Chapter 2 and Appendix B, only require inner products or kernel matrices rather than the original samples. For example, the sparse coding models use only inner product matrices: $\boldsymbol{K} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, $\boldsymbol{C} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$, and perhaps $\boldsymbol{R} = \boldsymbol{B}^{\mathrm{T}}\boldsymbol{B}$, where $\boldsymbol{A}$ and $\boldsymbol{B}$ represent the training data and unknown data with samples in columns, respectively. If the samples are normalized to have unit l_2-norm, the inner product of two samples is actually the cosine similarity between them. Suppose we have two unnormalized samples $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^m$. Their normalized inner product can be formulated as

$$\boldsymbol{a}'^{\mathrm{T}}\boldsymbol{b}' = \frac{\boldsymbol{a}^{\mathrm{T}}}{\|\boldsymbol{a}\|_2}\frac{\boldsymbol{b}}{\|\boldsymbol{b}\|_2}.$$

This feature of these optimization algorithms is quite useful to handle missing values. If $\boldsymbol{a}$ and $\boldsymbol{b}$ have missing values, we can use the features without missing values in both of them to calculate the cosine similarity:

$$\boldsymbol{a}'^{\mathrm{T}}\boldsymbol{b}' = \frac{\boldsymbol{a}_{\mathcal{I}}^{\mathrm{T}}}{\|\boldsymbol{a}_{\mathcal{I}}\|_2}\frac{\boldsymbol{b}_{\mathcal{I}}}{\|\boldsymbol{b}_{\mathcal{I}}\|_2}, \tag{8.4}$$

where $\mathcal{I}$ is the set of indices of the features whose values are observed in both $\boldsymbol{a}$ and $\boldsymbol{b}$, that is $\mathcal{I} = \mathcal{I}_a \cap \mathcal{I}_b$ where $\mathcal{I}_a$ is the set of indices of features whose values are present in $\boldsymbol{a}$. The difference between this weighting strategy with 0-imputation is that the inner product of a pair of 0-imputed samples is not the cosine similarity. If $\boldsymbol{a}$ and $\boldsymbol{b}$ are original samples with missing values imputed by 0, the normalized inner product of them is $\boldsymbol{a}'^{\mathrm{T}}\boldsymbol{b}' = \frac{\boldsymbol{a}_{\mathcal{I}}^{\mathrm{T}}}{\|\boldsymbol{a}_{\mathcal{I}_a}\|_2}\frac{\boldsymbol{b}_{\mathcal{I}}}{\|\boldsymbol{b}_{\mathcal{I}_b}\|_2}$.

Through this way, we can calculate the inner product matrices $\boldsymbol{K}$, $\boldsymbol{C}$, and $\boldsymbol{R}$ in a pair-wise fashion, if $\boldsymbol{A}$ and $\boldsymbol{B}$ have missing values. The difference between this weighting strategy with the global strategy of removing features with missing values as a preprocessing is that our strategy utilizes more features than the latter. This is because our strategy computes the inner product of two samples based on their available features, while the feature pre-removal strategy does so based on the global available features. In some cases, the missing

[3] This section is based on our publication [18].

rate can be so large that the majority of features are removed by the latter strategy. But our pair-wise weighting strategy might still work normally. Therefore, the advantage of our strategy is that it can work in the cases of large missing rate, as can be seen in our experiment presented in Section 4.2.

In the sparse coding models, if the nearest-subspace rule is used after getting the sparse coefficient of each sample, the regression residual of each class can also be computed in this weighting strategy. From the following equation,

$$\frac{1}{2}\|\boldsymbol{b}-\boldsymbol{A}_i\boldsymbol{x}_i\|_2^2=\frac{1}{2}(\boldsymbol{b}-\boldsymbol{A}_i\boldsymbol{x}_i)^{\mathrm{T}}(\boldsymbol{b}-\boldsymbol{A}_i\boldsymbol{x}_i)=\frac{1}{2}\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}-\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{b}+\frac{1}{2}\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{A}_i\boldsymbol{x}_i, \tag{8.5}$$

we can find that the residual of $\boldsymbol{b}$ regressed by the i-th class is a function of $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{b}$, $\boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{b}$ and $\boldsymbol{A}_i^{\mathrm{T}}\boldsymbol{A}_i$ which can be computed by the weighting strategy in the case of missing values. $\boldsymbol{A}_i$ is the training samples of the i-th class.

4.2 Experiment

For the purpose of exploring the capability of our weighting strategy of handling missing values, we did experiment on eight microarray data sets, as enumerated in Table 8.1, with randomly and artificially generated missing values. The missing rate (defined as the percentage of missing values out of a whole data set) ranges from 0.1 to 0.7. We used four-fold cross-validation for 50 runs at each missing rate for each data set. We compared our strategy with 0-imputation. The 0-imputation is a strategy to fill the missing values by constant 0's. We did not conduct KNNimpute, because for large missing rate, there is no complete feature and sample, and KNNimpute will thus fail for this case. This is one of the advantages of our pair-wise weighting strategy over the KNNimpute. We also involved the l_1LS (also known as SRC, see Chapter 2), *meta-sample based sparse representation classification* (MSRC) [48], *linear regression classification* (LRC) [109], 1-NN, SVM, and *extreme learning machine* (ELM) [238] using 0-imputation in the competition. We compared these methods on the microarray data sets as listed in Table 8.1. As an example, the average accuracies over SRBCT are plotted in Figure 8.1.

Furthermore, in order to investigate the robustness of our weighting strategy in the case of high missing rate, we employed the Friedman test (see Chapter 2) on all microarray data with missing rates 60% and 70% (therefore resulting in 16 data sets). We set the significance level to $\alpha = 0.05$. In our experiment, the null hypothesis was rejected, and the result of the Nemenyi test is shown in Figure 8.2.

First of all, we can see that NNLS-MAX using weighting strategy is significantly bet-

Table 8.1: Microarray data sets. The sample size of each class and the total data size are in the last column.

Data	#Class	#Feature	#Sample
Adenoma [69]	2	7457	18+18=36
Breast [70]	2	24481	44+34=74
Colon [44]	2	2000	40+22=62
Leukemia [72]	2	7129	47+25=72
ALL [77]	6	12625	15+27+64+20+43+79=248
Breast5 [31]	5	14460	13+22+53+31+13=158
MLL [79]	3	12582	24+20+28=72
SRBCT [80]	4	2308	23+8+12+20=63

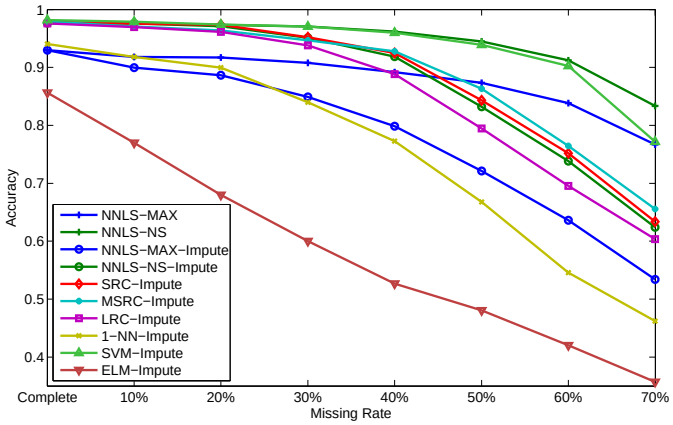


Figure 8.1: Classification performance on SRBCT as missing rate increases. This is a color figure, thus the readability may be affected if printed in grayscale.

ter than its counterpart using 0-imputation. If we increase the threshold of Type-I error slightly, then NNLS-NS using weighting strategy also performs significantly better than its counterpart using 0-imputation. Therefore it can be concluded that the weighting strategy is significantly better than 0-imputation when using the same classification technique. Secondly, from Figure 8.1 we can see that NNLS-NS using weighting strategy and SVM using 0-imputation still maintain high accuracy at severe missing rate. And from Figure 8.2, we can find both of them obtained top ranks. It hence can prove the robustness of our weighting strategy.

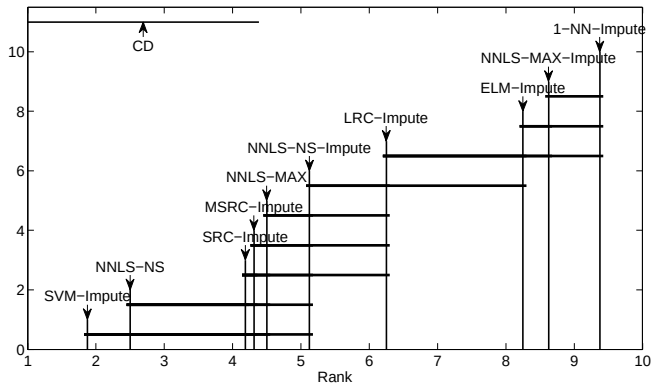


Figure 8.2: Graphical representation of Nemenyi test over microarray data with missing rates 60% and 70%.

5 The Extensions of KNNimpute and SVDimpute [4]

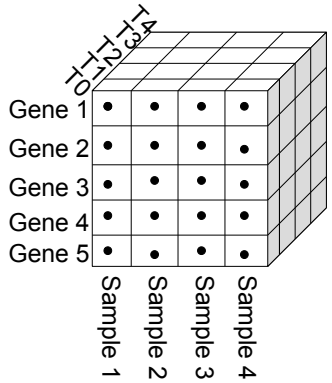
In this section, we modify the Average, KNNimpute and SVDimpute methods to pre-process incomplete GST data. In particular for the time-series in the GST data, we also investigate a pre-processing method that takes into account the temporal relationships within and between gene time-series.

In order to describe our methods clearly, we shall work on the data representations as illustrated in Figure 8.3. In the following subsections, m, n and p are respectively, the number of genes, samples, and time points in the GST data set which suffers from missing values; g_i, s_j and t_k are the gene, sample, and time point at row i, column j and time point k, respectively; g_{ij} is the time-series of gene g_i at sample s_j, or equivalently, s_{ij} is the time-series of sample s_j at gene g_i; g_{ijk} is a value at gene g_i, sample s_j and time point t_k; γ_{ijk} is the estimate of a missing value. A time-series is incomplete if it has a missing value. In the following, we assume that the gene time-series g_{ij} (i.e., sample time-series s_{ij}) has a missing value g_{ijk} at some time point t_k. In the data set we use, each gene g_i or sample s_j contains at least one complete time-series.

5.1 Average Methods

We implemented three average imputation methods, GAimpute (gene-average), SAimpute (sample-average) and GSAimpute (gene-sample-average). In GAimpute, the estimate γ_{ijk} of g_{ijk} is the average value at time point t_k of the m gene time-series of sample s_j. In

[4] This section is based on our publication [239].



(a) Tensor representation.

	Sample 1					Sample 2					Sample 3					Sample 4				
	T0	T1	T2	T3	T4	T0	T1	T2	T3	T4	T0	T1	T2	T3	T4	T0	T1	T2	T3	T4
Gene 1	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Gene 2	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Gene 3	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Gene 4	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Gene 5	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•

(b) Matrix representation.

Figure 8.3: Tensor representation (a) and matrix representation (b) of a gene-sample-time microarray data.

SAimpute, γ_{ijk} is the average of the n sample time-series of gene g_i at time point t_k. In GSAimpute, estimate γ_{ijk} of g_{ijk} is the average of the estimates obtained by GAimpute and SAimpute. These three methods are generically termed 3Aimpute, in subsequent sections.

5.2 k-Nearest Neighbor Methods

Similarly to the averaging methods above, we devised three KNNimpute methods on incomplete gene or sample time-series. For each incomplete time-series g_{ij} or s_{ij}, k nearest neighbors are computed only from complete time-series. The estimate γ_{ijk} of g_{ijk} is then taken as the weighted average at time point t_k of these nearest neighbors.

In GKNNimpute, the k neighbors of g_{ij} are obtained from all the complete gene time-series of sample s_j. In SKNNimpute, the neighbors are found from all the complete sample time-series of gene g_i. When the samples have class labels, then the k complete sample time-series must be of the same class as the incomplete time-series s_{ij}. In GSKNNimpute, γ_{ijk} is the average of the estimates from GKNNimpute and SKNNimpute. We denote these

methods by the generic term 3KNNimpute.

For missing value estimation, the Euclidean distance performed better than other metrics in [108] and [230]; hence we have used this metric to find the nearest neighbors. We have also used the novel *integral-distance* metric, proposed in [192, 168, 184, 180], which was shown to be more robust than the Pearson correlation distance on time-series data. In term of notation, when using the Euclidean distance, we will add letter "E" after KNN, otherwise we add letter "A".

5.3 k-Nearest Neighbor with Multiple Time-Series Alignment

The 3KNNAimpute methods based on the integral-distance function are quite slow due to computing the pair-wise alignment between two time-series, when computing their distance. Pair-wise alignment was introduced in [180] and [192] in order to take into account the temporal relationships between time-series and within time-series, and thus to perform a more robust analysis of time-series data. *Multiple time-series alignment* (MA) was introduced in [192] as a pre-processing stage for clustering gene time-series, and was shown to yield much faster clustering time. MA is a series of transformations performed on all time-series profiles, at once, such that the area between any two transformed profiles is minimal, and thus the subsequent clustering method needs not to apply pair-wise alignment for computing the integral-distance between time-series profiles.

For imputation with MA (see Figure 8.4), we first apply GKNNimpute, with the Euclidean distance, on the original data M in order to obtain a complete data set C; this is because MA requires a complete data set. We then perform the MA transformations of [192] on C to yield a new data set A. For each incomplete time-series g_{ij} in M with missing value g_{ijk} at t_k, we find the k nearest neighbors of its counterpart in A using integral-distance (no alignment is necessary here). We then estimate g_{ijk} in A as the weighted average of these neighbors at time point t_k. This is essentially GKNNimpute applied on A, except that the nearest neighbors in A must be complete in M. After imputing g_{ijk} in A, we apply the inverse transformation MA^{-1} [192] to obtain the estimation of g_{ijk} in M. SKNNimpute and GSKNNimpute can also be applied in a similar way. We use the generic term 3MAimpute for methods based on MA.

5.4 Singular Value Decomposition

We modified the SVDimpute method of [240] for GST data. First, SVD is performed on all complete time-series of gene g_i or sample s_j to find a set of k eigen-samples or eigen-genes. Each incomplete gene or sample time-series is then a linear combination of

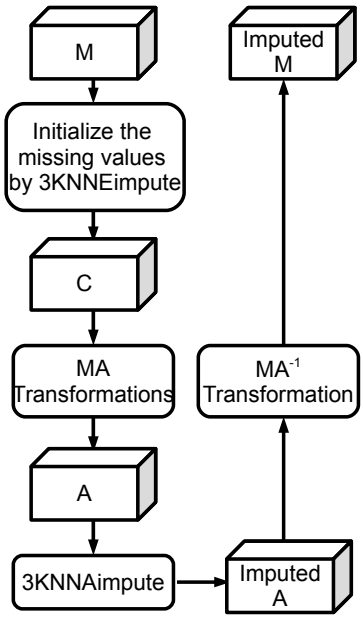


Figure 8.4: k-NN imputation with multiple-alignment.

these eigenvectors. We apply linear least square regression to obtain the coefficients of these eigenvectors for each incomplete time-series. Missing values are then imputed by linear combination of the corresponding values of the eigenvectors. These methods are known as 3SVDimpute, including: GSVDimpute for imputing on sample s_j at time point t_k, SSVDimpute for imputing on gene g_i at time point t_k, and the combined GSSVDimpute.

5.5 Time-Point Estimation for GST Data

In some GST data sets, a sample s_j may have a missing value at time point t_k for all its gene time-series g_{ij}. Thus the entire time point t_k is missing for that sample. Most existing imputation algorithms on two-way microarray data, such as Average, KNNimpute and SVDimpute, are not able to deal with this situation. An exception is ARLSimpute [232]. Assume sample s_j, has missing values g_{1jk}, g_{2jk}, $\cdots$, g_{mjk} at time point t_k (in all its gene time-series g_{ij}). Our G*impute methods cannot estimate these values. However, our S*impute methods have no such limitation: since an incomplete gene time-series g_{ij} of s_j will be imputed from the other complete sample time-series s_{ij} of g_i (since g_i contains at least one complete sample time-series).

5.6 Evaluation of Imputation Methods

Let $G = [g_{ijk}]$ be a complete GST data set and $H = [h_{ijk}]$ be an incomplete data set obtained from G by removing some values at random. Let $\Gamma = [\gamma_{ijk}]$ be an imputation of H: that is, obtained after applying an imputation algorithm on H. We can assess the performance of an imputation method by measuring the error $E(G;\Gamma)$ between G and Γ. We use the *normalized root mean squared* (NRMS) error defined as:

$$E(G;\Gamma) = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p}(g_{ijk}-\gamma_{ijk})^2}{\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{p}(g_{ijk})^2}}, \tag{8.6}$$

where $\{m, n, p\}$ is the size of G. When G is known, then the missing value imputation problem is a supervised learning problem. Most microarray data sets are initially incomplete; thus $G = H$, and therefore, the missing value imputation problem becomes an unsupervised learning problem to find a true completion of G. In the next section, we test our algorithms only on complete data sets G. That is, given an initial but incomplete GST data set M, we first remove from M every gene, sample and time point containing missing values, and hence, producing a complete sub-structure G of M. We then generate a new data set H from G by randomly deleting some values from G (G and H have the same dimensions). Next, we perform an imputation on H to obtain a complete Γ and compute the error $E(G, \Gamma)$. We repeat this process 100 times randomly altering G then computing Γ. In the experiments below, we report the average error over 100 iterations. See Figure 8.5 for a sketch of the experimental procedure.

5.7 Experiment

In this study, we used the interferon-β (IFNβ) data set of [112]. As we mentioned in the previous chapters, this GST data contains 76 genes, 53 samples (from 31 good responders and 22 bad responders), and 7 time points (at 0, 3, 6, 9, 12, 18, 24 months). 10.55% of expression intensities and 36 time points (one complete sample has 7 time points) are missing. There are 26 samples containing at least one missing time point. The samples are taken from patients suffering with relapsing-remitting multiple sclerosis and treated with IFNβ as initial therapy. Their blood samples are obtained by veni-puncture at each time point to produce the microarray data. After two years, the patients are classified as either good or bad responders according to strict experimental criteria. We performed our experiments on this data set using the experimental procedure described in Figure 8.5, to evaluate our imputations algorithms. Here, M is the original incomplete IFNβ data set,

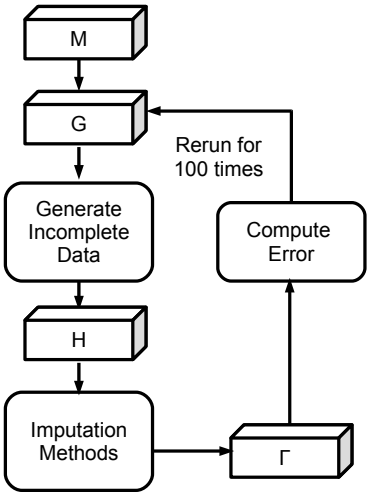


Figure 8.5: Experimental procedure of imputation.

and removing incomplete genes, samples and time points resulted in a complete data set G containing 53 genes, 27 samples (18 good and 9 bad responders) and 7 time points. All of the approaches are implemented in MATLAB.

The incomplete data set H is obtained from G as follows: we randomly select $g\%$ of all genes time-series in G to be altered, then for each such time-series g_{ij}, we randomly remove at most t values; we randomly select a sample s_j, then remove all values at time point t_k, that is we delete the time point t_k for some arbitrary k. After performing an imputation method on H we obtain the estimated Γ, then compute the error $E(G, \Gamma)$.

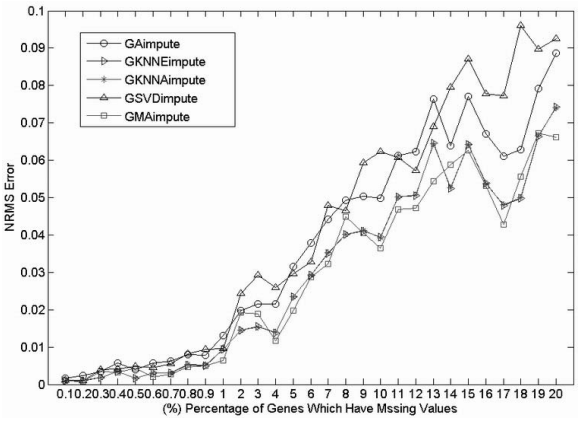
Table 8.2 shows for each method, the number of times when it has produced the lowest estimation error for $g\% = 0.1\%, 0.2\%, \cdots, 0.9\%, 1\%, 2\%, \cdots$, and 20% (that is for 29 values of $g\%$) when at most $t = 1, 2, 3, 4, 5$ values are missing in the selected gene time-series (columns of the table). Both small and large values of $g\%$ are used because we want to investigate the performances of the extended methods at small and large numbers of missing values. In Table 8.2, we did not remove any time point. As shown in the table, GMAimpute is the best method among the G*impute methods (which use the gene time-series from the same samples to estimate the missing values), while SKNNAimpute is the best method among the S*impute methods (that use the sample time-series to estimate missing values). For instance, for $t = 4$, GMAimpute outperforms the other G*impute methods for 25 times out of 29 value of $g\%$. SKNNAimpute outperforms the other S*impute methods for all values of t.

Table 8.2: Statistics for the five methods for 29 different percentages (from 0.1% to 20%) of genes which contain at most 1, 2, 3, 4, or 5 missing values, respectively. There is no time point missing in this case.

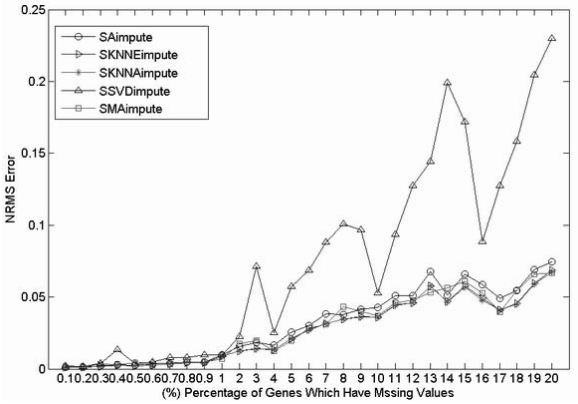
Method	At Most				
	1	2	3	4	5
GAimpute	0	0	0	0	0
GKNNEimpute	5	8	8	3	6
GKNNAimpute	3	5	0	1	1
GSVDimpute	0	0	0	0	0
GMAimpute	21	16	21	25	22
SAimpute	0	0	1	0	0
SKNNEimpute	2	5	7	4	3
SKNNAimpute	17	15	12	10	15
SSVDimpute	0	0	0	0	0
SMAimpute	10	9	9	15	11
GSAimpute	0	0	0	0	0
GSKNNEimpute	4	5	4	7	3
GSKNNAimpute	16	13	12	5	12
GSSVDimpute	0	0	0	0	0
GSMAimpute	9	11	13	17	14

Figure 8.6a shows the performance of G*impute methods for $t = 1$. The horizontal axis presents various values of $g\%$, and the vertical axis stands for the NRMS error at respective $g\%$. Clearly, GMAimpute gives the lowest error on average as $g\%$ increases. In general for all values of t, we found that KNN-based methods yield the best results while SVD-based methods give the worst results. Additionally, as $g\%$ increases, the NRMSs of all algorithms climb up gradually. KNN methods use local information around an incomplete time-series for estimation: i.e., a subset of time-series similar to the given incomplete time-series. The averaged methods estimate an incomplete time-series by the centroid of either a gene or a sample, but unlike in KNN, the centroid may be very far from the incomplete time-series to estimate. In our data set however, SVD performed worst simply because there are not enough complete time-series, and thus is outperformed by average method. In general in all our experiments with G*impute, S*impute and GS*impute, SVD underperformed all other methods. See Figures 8.6b and 8.6c for instance, for increasing $g\%$ with $t = 1$, where the NRMS error of SVD based methods severely fluctuate and dramatically deteriorate.

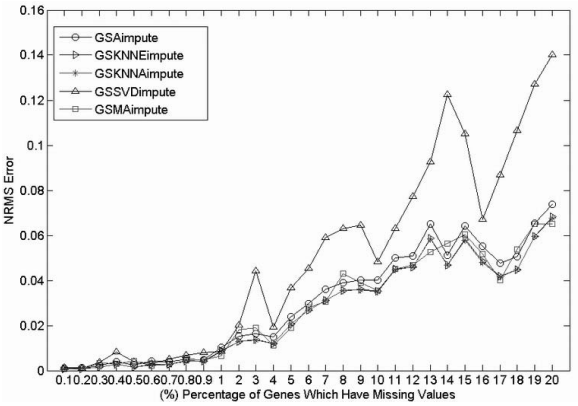
We also performed experiments for the case where data set H contains a sample with one time point that is entirely missing. Tables 8.3, 8.4, and 8.5 show the NRMS error of G*impute, S*impute, and GS*impute methods, respectively, when one time point missing



(a) G*impute methods.



(b) S*impute methods.



(c) GS*impute methods.

Figure 8.6: Errors of G*impute (a), S*impute (b), and GS*impute (c) methods, for $g\% =$ 0.1% to 20% and $t = 1$.

and at most $t = 1$ value missing for each of the $g\%$ (the first column) selected time-series. In all the three tables, the best result of each row is in bold. In Table 8.3, GKNNAimpute outperforms all the other G*impute methods in 17 out of 29 values of $g\%$. Likewise in Tables 8.4 and 8.5, SKNNAimpute and GSKNNAimpute outperform their respective competitors. Thus, the KNN-based methods are the best in all experiments.

Table 8.3: Mean NRMS error for G*impute with at most one missing value for each selected time-series and one missing time point.

Percent	GAimpute	GKNNEimpute	GKNNAimpute	GMAimpute
0.1%	0.0394	0.0338	**0.0330**	0.0360
0.2%	0.0346	0.0270	0.0271	**0.0249**
0.3%	0.0386	0.0329	0.0330	**0.0329**
0.4%	0.0276	0.0237	**0.0235**	0.0279
0.5%	0.0333	0.0251	**0.0249**	0.0314
0.6%	0.0365	0.0308	**0.0308**	0.0315
0.7%	0.0433	0.0377	0.0380	**0.0358**
0.8%	0.0293	0.0248	**0.0247**	0.0303
0.9%	0.0370	0.0285	0.0287	**0.0277**
1%	0.0344	0.0265	**0.0259**	0.0262
2%	0.0541	0.0470	0.0466	**0.0425**
3%	0.0358	0.0292	**0.0288**	0.0425
4%	0.0517	0.0435	**0.0429**	0.0479
5%	0.0439	0.0341	**0.0339**	0.0394
6%	0.0435	0.0350	0.0351	**0.0346**
7%	0.0465	0.0379	**0.0374**	0.0430
8%	0.0591	**0.0484**	0.0486	0.0531
9%	0.0641	0.0522	**0.0521**	0.0533
10%	0.0660	0.0546	**0.0540**	0.0591
11%	0.0792	0.0666	**0.0661**	0.0706
12%	0.0799	0.0662	**0.0660**	0.0735
13%	0.0780	0.0664	**0.0659**	0.0664
14%	0.0707	0.0565	0.0569	**0.0553**
15%	0.0860	**0.0717**	0.0719	0.0735
16%	0.0774	0.0632	**0.0627**	0.0636
17%	0.0893	0.0766	0.0764	**0.0706**
18%	0.0821	**0.0687**	0.0690	0.0821
19%	0.0961	0.0811	**0.0809**	0.0823
20%	0.1037	0.0903	0.0894	**0.0881**

In all experiments, GMAimpute performed the best with small g% and no missing time points. MA requires a complete data set for transformation, and hence, the initialization

Table 8.4: Mean NRMS error for S*impute with at most one missing value for each selected time-series and one missing time point.

Percent	SAimpute	SKNNEimpute	SKNNAimpute	SMAimpute
0.1%	0.0393	0.0338	**0.0330**	0.0360
0.2%	0.0343	**0.0272**	0.0273	0.0250
0.3%	0.0382	**0.0326**	0.0327	0.0331
0.4%	0.0266	0.0231	**0.0230**	0.0277
0.5%	0.0335	0.0250	**0.0247**	0.0315
0.6%	0.0363	**0.0307**	0.0308	0.0315
0.7%	0.0428	0.0375	0.0380	**0.0356**
0.8%	0.0282	0.0247	**0.0245**	0.0302
0.9%	0.0368	0.0280	0.0283	**0.0276**
1%	0.0340	0.0263	**0.0257**	0.0261
2%	0.0517	0.0457	0.0455	**0.0427**
3%	0.0331	0.0277	**0.0273**	0.0412
4%	0.0494	0.0442	**0.0433**	0.0486
5%	0.0410	0.0320	**0.0315**	0.0387
6%	0.0380	**0.0324**	0.0329	0.0337
7%	0.0424	0.0355	**0.0348**	0.0424
8%	0.0532	0.0472	**0.0470**	0.0534
9%	0.0584	0.0514	**0.0507**	0.0553
10%	0.0624	0.0526	**0.0520**	0.0592
11%	0.0719	0.0634	**0.0623**	0.0686
12%	0.0738	0.0649	**0.0646**	0.0728
13%	0.0713	**0.0615**	0.0616	0.0656
14%	0.0610	**0.0497**	0.0499	0.0537
15%	0.0739	0.0648	**0.0642**	0.0726
16%	0.0665	0.0571	**0.0563**	0.0621
17%	0.0790	0.0695	0.0694	**0.0690**
18%	0.0727	**0.0645**	**0.0645**	0.0822
19%	0.0850	0.0759	**0.0757**	0.0811
20%	0.0927	0.0804	**0.0786**	0.0844

process of Figure 8.4 will essentially affect the performance of MA-based imputation methods. For large value of $g\%$, such initialization would lead to more imprecise alignment which will eventually deteriorate the estimation of missing values. The 3KNNAimpute methods, which use the integral-distance of [180] and [192], outperform the KNN methods based on the Euclidean distance; in most experiments (see the tables 8.3, 8.4, and 8.5). In the context of time-series analysis, it has been shown in [180] that the integral-distance is more robust than the Pearson correlation distance. It remains to be investigated why the

Table 8.5: Mean NRMS error for GS*impute with at most one missing value for each selected time-series and one missing time point.

Percent	GSAimpute	GSKNNEimpute	GSKNNAimpute	GSMAimpute
0.1%	0.0393	0.0338	**0.0329**	0.0360
0.2%	0.0344	0.0271	0.0272	**0.0249**
0.3%	0.0382	**0.0327**	0.0328	0.0330
0.4%	0.0268	0.0234	**0.0232**	0.0277
0.5%	0.0331	0.0249	**0.0247**	0.0314
0.6%	0.0360	**0.0307**	**0.0307**	0.0314
0.7%	0.0425	0.0375	0.0379	**0.0356**
0.8%	0.0282	0.0245	**0.0244**	0.0301
0.9%	0.0363	0.0281	0.0283	**0.0276**
1%	0.0338	0.0263	**0.0256**	0.0261
2%	0.0516	0.0460	0.0456	**0.0425**
3%	0.0328	0.0278	**0.0272**	0.0416
4%	0.0483	0.0429	**0.0422**	0.0479
5%	0.0394	0.0319	**0.0317**	0.0387
6%	0.0378	**0.0325**	0.0327	0.0336
7%	0.0412	0.0355	**0.0348**	0.0421
8%	0.0525	0.0466	**0.0465**	0.0528
9%	0.0573	0.0503	**0.0499**	0.0538
10%	0.0585	0.0517	**0.0510**	0.0584
11%	0.0714	0.0636	**0.0628**	0.0691
12%	0.0724	0.0638	**0.0634**	0.0725
13%	0.0697	0.0623	**0.0621**	0.0653
14%	0.0607	**0.0507**	0.0509	0.0535
15%	0.0747	0.0661	**0.0659**	0.0722
16%	0.0661	0.0582	**0.0574**	0.0620
17%	0.0786	0.0711	0.0710	**0.0692**
18%	0.0709	**0.0638**	0.0640	0.0810
19%	0.0844	0.0758	**0.0754**	0.0805
20%	0.0915	**0.0831**	**0.0813**	0.0854

integral-distance performed better than the Euclidean distance. 3KNNAimpute has the advantage of estimating both missing values and missing time points. The Euclidean distance does not consider the order of the time points, i.e. exchanging the order of the time points will result in the same distance. The integral-distance takes into account the order of time points. Exchanging the time points would likely produce different distances. Moreover, the area based distance measure can find similar genes which might be dissimilar in the context of the Euclidean distance. The 3MAimpute methods outperform the 3KNNEimpute

algorithms for small values of $g\%$.

6 Conclusions

In this chapter, we have focused on discussing the missing value issues in two-way and three-way data. We have surveyed the three strategies of handling missing values. We then proposed our pair-wise weighting strategy to compute inner products or kernel matrices for kernel models. After that, we extended the successful imputation methods, originally invented for two-way data, to the versions for three-way data. One thing we need to mention, at this moment, the principle that no method is generally better than another method. We have to select a proper method for our specific application and specific data. If there are sufficient samples and only few samples have missing values, then we may simply remove these samples. If the number of available samples is very limited, and the missing rate is small, then we may choose estimation methods. If there is a large missing rate, we may use a weighting method. If an application requires a fast response, then we have to choose a fast method to deal with missing values.

Publications

1. We extended the existing imputation methods, originally for two-way data, to methods for gene-sample-time data in [*Y. Li, A. Ngom, and L. Rueda, "Missing value imputation methods for gene-sample-time microarray data analysis," IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), Montreal, Canada, May 2010, pp.183-189.*].

2. We proposed a pair-wise weighting method for computing kernel matrices from incomplete data in [*Y. Li and A. Ngom, "Classification approach based on non-negative least squares," Neurocomputing, vol. 118, pp. 41-57, 2013.*].

Appendix A

An Introduction to Tensor Algebra

In this appendix, we briefly introduce the main concepts in *tensor algebra*, which is very necessary to help readers to understand the methodologies presented Chapter 4.

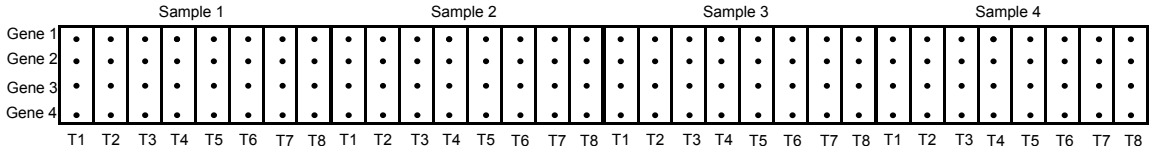
1 Notations and Tensor Manipulations

Hereafter, we use the following notations unless otherwise noted:

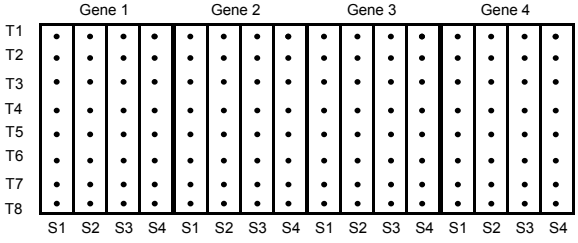
- A matrix is denoted by a bold capital letter, e.g. $\boldsymbol{A}$.
- A (column) vector is denoted by a bold lowercase letter, e.g. $\boldsymbol{a}$.
- A bold lowercase letter with a subscript $\boldsymbol{a}_i$ denotes the i-th column vector in matrix $\boldsymbol{A}$.
- The italic lowercase letter with two subscripts a_{ij} is the (i, j)-th scalar element of matrix $\boldsymbol{A}$.
- A boldface Euler script, e.g. $\boldsymbol{\mathcal{X}}$, denotes a three (or higher)-order tensor, for example, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$.
- $\boldsymbol{X}_{(1)p}$ denotes the p-th frontal slice of $\boldsymbol{\mathcal{X}}$, of size $I_1 \times I_2$.
- $\boldsymbol{X}_{(n)}$ denotes the matrix obtained through the mode-n matricization of the tensor $\boldsymbol{\mathcal{X}}$. Columns of $\boldsymbol{X}_{(n)}$ are the mode-n fibers of tensor $\boldsymbol{\mathcal{X}}$. A mode-n fiber is a vector defined through fixing every index but the n-th index. This is the extension of matrix row and column in tensor algebra. $\boldsymbol{X}_{(1)}$ therefore denotes the matrix of size $I_1 \times (I_2 \times I_3)$, unfolded in mode-1 of $\boldsymbol{\mathcal{X}}$, that is $\boldsymbol{X}_{(1)} = [\boldsymbol{X}_{(1)1}, \boldsymbol{X}_{(1)2}, \cdots, \boldsymbol{X}_{(1)I_3}]$, where $\boldsymbol{X}_{(1)i_3}$ is a

slice of the third axis. Similarly, $\boldsymbol{X}_{(2)} = [\boldsymbol{X}_{(2)1}, \boldsymbol{X}_{(2)2}, \cdots, \boldsymbol{X}_{(2)I_1}]$ of size $I_2 \times (I_3 \times I_1)$ is the mode-2 matricization of the tensor $\boldsymbol{\mathcal{X}}$. $\boldsymbol{X}_{(3)} = [\boldsymbol{X}_{(3)1}, \boldsymbol{X}_{(3)2}, \cdots, \boldsymbol{X}_{(3)I_2}]$ of size $I_3 \times (I_1 \times I_2)$ is the mode-3 matricization of the tensor $\boldsymbol{\mathcal{X}}$. In Figure A.1a, each column is a mode-1 fiber obtained via fixing the sample and time axes. Each such fiber is the gene profiles of a specific sample at a specific time point. The matrix in Figure A.1a, is obtained by placing all mode-1 fibers together in the columns, that is concatenating the *sample slices* along the column direction. Similarly, the matrix in Figure A.1b is the mode-2 matricization, and is obtained by placing all mode-2 fibers together in the columns, that is concatenating the *gene slices* along the column direction. The matrix in Figure A.1c is the mode-3 matricization, and is obtained by placing all mode-3 fibers together in the columns, that is concatenating the *time slices* along the column direction.

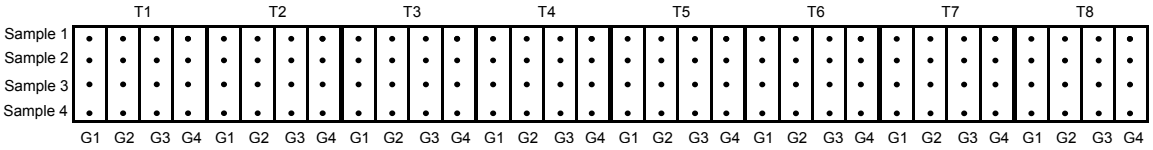
- The (i_1, i_2, i_3)-th scalar element of $\boldsymbol{\mathcal{X}}$ is denoted by $x_{i_1 i_2 i_3}$.



(a) Mode-1 matricization.



(b) Mode-2 matricization.



(c) Mode-3 matricization.

Figure A.1: The mode-1 (a), mode-2 (b), and mode-3 (c) matricizations of the three-way tensor shown in Figure 4.1.

Suppose that $\boldsymbol{a} \in \mathbb{R}^m$, $\boldsymbol{b} \in \mathbb{R}^n$, and $\boldsymbol{c} \in \mathbb{R}^r$. The operator "$\circ$" in $\boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{M} \in \mathbb{R}^{m \times n}$ is vector outer product. $\boldsymbol{M}$ is a rank-one matrix. Its elements can be computed as $m_{ij} = a_i b_j$.

Similarly, the operation $\boldsymbol{a} \circ \boldsymbol{b} \circ c = \boldsymbol{\mathcal{M}} \in \mathbb{R}^{m \times n \times r}$, where $\boldsymbol{\mathcal{M}}$ is a rank-one tensor and $m_{ijk} = a_i b_j c_k$.

Suppose that $\boldsymbol{A} \in \mathbb{R}^{m_1 \times n_1}$ and $\boldsymbol{B} \in \mathbb{R}^{m_2 \times n_2}$, their Kronecker product is defined as

$$\begin{aligned} \boldsymbol{A} \otimes \boldsymbol{B} &= \begin{bmatrix} a_{11}\boldsymbol{B} & a_{12}\boldsymbol{B} & \cdots & a_{1n_1}\boldsymbol{B} \\ a_{21}\boldsymbol{B} & a_{22}\boldsymbol{B} & \cdots & a_{2n_1}\boldsymbol{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m_1 1}\boldsymbol{B} & a_{m_1 2}\boldsymbol{B} & \cdots & a_{m_1 n_1}\boldsymbol{B} \end{bmatrix} \\ &= [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \boldsymbol{a}_1 \otimes \boldsymbol{b}_2, \cdots, \boldsymbol{a}_{n_1} \otimes \boldsymbol{b}_{n_2-1}, \boldsymbol{a}_{n_1} \otimes \boldsymbol{b}_{n_2}]. \end{aligned} \tag{A.1}$$

We have that $\boldsymbol{A} \otimes \boldsymbol{B} \in \mathbb{R}^{(m_1 m_2) \times (n_1 n_2)}$.

Suppose $\boldsymbol{A} \in \mathbb{R}^{m_1 \times n}$ and $\boldsymbol{B} \in \mathbb{R}^{m_2 \times n}$, their Khatri-Rao product is defined as

$$\boldsymbol{A} \odot \boldsymbol{B} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1, \boldsymbol{a}_2 \otimes \boldsymbol{b}_2, \cdots, \boldsymbol{a}_{n-1} \otimes \boldsymbol{b}_{n-1}, \boldsymbol{a}_n \otimes \boldsymbol{b}_n]. \tag{A.2}$$

Therefore $\boldsymbol{A} \odot \boldsymbol{B}$ is of size $(m_1 m_2) \times n$.

The Hadamard product is also called element-wise product which is denoted as $\boldsymbol{A} * \boldsymbol{B}$.

The n-mode product of a tensor $\boldsymbol{\mathcal{X}}$ and a matrix $\boldsymbol{A}$, written as $\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}$, is:

$$[\boldsymbol{\mathcal{X}} \times_n \boldsymbol{A}]_{i_1 \times \cdots i_{n-1} \times j \times i_{n+1} \times \cdots \times i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} a_{j i_n} \ , \tag{A.3}$$

where $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and $\boldsymbol{A} \in \mathbb{R}^{J \times I_n}$. This results in a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N}$.

$\boldsymbol{\mathcal{X}}$ can be matricized into matrices in different modes. For example,
$\boldsymbol{X}_{(1)} = [\boldsymbol{X}_{(1)1}, \boldsymbol{X}_{(1)2}, \cdots, \boldsymbol{X}_{(1)I_3}]$ is matricized in the first mode (see Figure A.1).

2 Tensor Decomposition

Tensor decomposition methods mainly include PARAFAC and Tucker decompositions [125]. PARAFAC is the abbreviation of *parallel factors*. It is also called *canonical decomposition* (CANDECOMP). It factorizes a tensor into a summation of rank-one tensors. Suppose that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$, PARAFAC decomposes $\boldsymbol{\mathcal{X}}$ into

$$\boldsymbol{\mathcal{X}} \approx \sum_{r=1}^{R} \boldsymbol{g}_r \circ \boldsymbol{t}_r \circ \boldsymbol{s}_r, \tag{A.4}$$

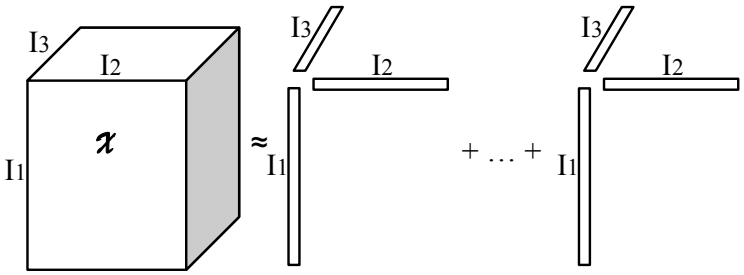


Figure A.2: PARAFAC decomposition.

where R is the rank of the reconstructed tensor, $\boldsymbol{g}_r \in \mathbb{R}^{I_1}$, $\boldsymbol{t}_r \in \mathbb{R}^{I_2}$, and $\boldsymbol{s}_r \in \mathbb{R}^{I_3}$ are columns of $\boldsymbol{G}$, $\boldsymbol{T}$, and $\boldsymbol{S}$, respectively. This factorization can be concisely written as $\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}]\!]$, where $\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}$ are called *factor matrices*. The approximation $\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}]\!]$ can be matricized into different modes:

$$\begin{cases} \boldsymbol{X}_{(1)} \approx \boldsymbol{G}(\boldsymbol{S} \odot \boldsymbol{T})^{\mathrm{T}} \\ \boldsymbol{X}_{(2)} \approx \boldsymbol{T}(\boldsymbol{S} \odot \boldsymbol{G})^{\mathrm{T}} \\ \boldsymbol{X}_{(3)} \approx \boldsymbol{S}(\boldsymbol{T} \odot \boldsymbol{G})^{\mathrm{T}} \end{cases} . \tag{A.5}$$

Figure A.2 is an illustration of PARAFAC decomposition.

PARAFAC can be optimized by *alternating least squares* (ALS) algorithm [234]. If non-negativity is constrained on all factor matrices, *alternating non-negative least squares* (ANLS) algorithm can be used [234].

The Tucker3 model of Tucker decomposition factorizes a tensor $\boldsymbol{\mathcal{X}}$ into a core tensor $\mathcal{C}$ and three mode matrices $\boldsymbol{G}$, $\boldsymbol{T}$, and $\boldsymbol{S}$ as follows:

$$\boldsymbol{\mathcal{X}} \approx \mathcal{C} \times_1 \boldsymbol{G} \times_2 \boldsymbol{T} \times_3 \boldsymbol{S} = \sum_{j_1=1}^{J_1} \sum_{j_2=1}^{J_2} \sum_{j_3=1}^{J_3} c_{j_1 j_2 j_3} \boldsymbol{g}_{j_1} \circ \boldsymbol{t}_{j_2} \circ \boldsymbol{s}_{j_3} = [\![\mathcal{C}; \boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}]\!] , \tag{A.6}$$

where $\mathcal{C}$ is a core tensor and $\boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}$ are called mode matrices. The decomposition is illustrated in Figure A.3. In light of Equation (A.6), it is clear that an element of core tensor $\mathcal{C}$ indicates the degree of interaction among the corresponding mode vectors from different mode matrices. For instance, $c_{j_1 j_2 j_3}$ reflects the interaction between $\boldsymbol{g}_{j_1}$, $\boldsymbol{t}_{j_2}$, and $\boldsymbol{s}_{j_3}$. Applying matricization on both sides of $\boldsymbol{\mathcal{X}} \approx [\![\mathcal{C}; \boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}]\!]$, we can have the following

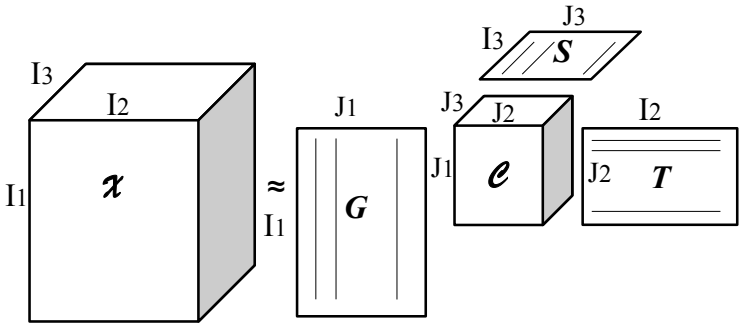


Figure A.3: Tucker3 decomposition.

relations:

$$\begin{cases} \boldsymbol{X}_{(1)} \approx \boldsymbol{G}\boldsymbol{C}_{(1)}(\boldsymbol{S} \otimes \boldsymbol{T})^{\mathrm{T}} \\ \boldsymbol{X}_{(2)} \approx \boldsymbol{T}\boldsymbol{C}_{(2)}(\boldsymbol{S} \otimes \boldsymbol{G})^{\mathrm{T}} \\ \boldsymbol{X}_{(3)} \approx \boldsymbol{S}\boldsymbol{C}_{(3)}(\boldsymbol{T} \otimes \boldsymbol{G})^{\mathrm{T}} \end{cases}, \tag{A.7}$$

where $\boldsymbol{X}_{(1)}$ and $\boldsymbol{G}\boldsymbol{C}_{(1)}(\boldsymbol{S} \otimes \boldsymbol{T})^{\mathrm{T}}$ are obtained through matricizing $\mathcal{X}$ and $[\![\mathcal{C}; \boldsymbol{G}, \boldsymbol{T}, \boldsymbol{S}]\!]$ in mode 1, respectively.

Generally speaking, there are no constraints on the core tensor and mode matrices in Tucker decomposition. However, constraints such as orthogonality, non-negativity, and non-Gaussianity can be enforced in a decomposition algorithm. For instance, HOSVD enforces the orthogonality constraints on the mode matrices and is among the most popular Tucker algorithms. The mode matrices can be obtained by applying SVD on the matricized matrices of the corresponding mode. In Equation (A.7), we can apply SVD on $\boldsymbol{X}_{(1)}$: $\boldsymbol{X}_{(1)} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$, and $\boldsymbol{G}$ can then be obtained by taking the J_1 leading singular vectors of $\boldsymbol{U}$. Similarly, $\boldsymbol{T}$ and $\boldsymbol{S}$ can be obtained by applying SVD on $\boldsymbol{X}_{(2)}$ and $\boldsymbol{X}_{(3)}$, respectively. The core tensor is obtained through $\mathcal{C} = \mathcal{X} \times_1 \boldsymbol{G}^T \times_2 \boldsymbol{T}^T \times_3 \boldsymbol{S}^T$. Interested readers are referred to [241] for more details. *High-order orthogonal iterations* (HOOI) is an alternating least squares (ALS) algorithm initialized by HOSVD, which gives better decomposition than HOSVD itself (see [242] and [125] for details). HOOI also generates orthogonal mode matrices. HONMF imposes non-negativity constraints on the core tensor and mode matrices. Multiplicative updates rules [12] corresponding to core and mode matrices have been extended in [123]. The core tensor and mode matrices are alternatingly updated until the convergence criteria are met. The authors of [12] have observed that good interpretation and learning performance can be benefited by adding non-negativity and sparsity constraints on matrix factorization.

Even though the sparsity can be imposed and controlled. Sparsity is sometimes a by-product of non-negativity constrained matrix (maybe tensor also) factorization without explicit sparsity constraint.

Appendix B

An Introduction to Regularized Linear Models

1 Background

Many problems in bioinformatics can be formulated into classification and regression problems in machine learning. Given data $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n]$, and the response $\boldsymbol{y} = [y_1; y_2; \cdots; y_n]$, the process of learning a model, say f, is to minimize the following *expected risk* [243] with is the expectation of the loss with respect to the joint probability distribution $p(\boldsymbol{x}, y)$:

$$R[f] = E[f] = \int l(f, \boldsymbol{x}, y) dp(\boldsymbol{x}, y), \tag{B.1}$$

where $l(f, \boldsymbol{x}, y)$ is a *loss function*, and $p(\boldsymbol{x}, y)$ is the joint density function. The most popular loss functions are given as below:

$$l(f, \boldsymbol{x}, y) = \begin{cases} \max\{0, |y - f(\boldsymbol{x})| - \varepsilon\} = |y - f(\boldsymbol{x})|_\varepsilon & \varepsilon\text{-sensitive loss} \\ \max(0, 1 - yf(\boldsymbol{x})) = |1 - yf(\boldsymbol{x})|_+ & \text{hinge loss} \\ (y - f(\boldsymbol{x}))^2 & \text{squared loss} \\ |y - f(\boldsymbol{x})| & l_1\text{-loss} \\ \log(1 + e^{-yf(\boldsymbol{x})}) & \text{logistic loss} \end{cases}. \tag{B.2}$$

Unfortunately, in most cases, we can not directly apply this formulation, because we do not know the actual joint distribution $p(\boldsymbol{x}, y)$. Minimizing the following *empirical error*

only can overfit the data, and therefore does not imply a good capability of generalization:

$$R_{emp}[f|\boldsymbol{X},\boldsymbol{y}] = \frac{1}{n}\sum_{i=1}^{n} l(f,\boldsymbol{x}_i,y_i). \tag{B.3}$$

A typical example of minimizing empirical error is the least-squares method for learning a linear model for classification and regression. A linear model can be expressed as

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b, \tag{B.4}$$

where $\boldsymbol{w}$ and b are model parameters: $\boldsymbol{w}$ is the normal vector of the hyperplane, and b is the bias term. For classification, the decision function is an indicator function:

$$d(\boldsymbol{x}) = \mathrm{sign}[f(\boldsymbol{x}|\boldsymbol{w}^*,b^*)], \tag{B.5}$$

where $\{\boldsymbol{w}^*,b^*\}$ is the optimal parameter learned. The least-squares method learns the parameters by the following optimization task:

$$\frac{1}{2}\|\boldsymbol{y} - (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{b})\|_2^2, \tag{B.6}$$

where $\boldsymbol{b} = \{b\}^n$, that is a constant column vector with values b. It is well-known that the least-squares method has poor capability of generalization [244].

In order to avoid overfitting, we need to control the model complexity. That is, we need a trade-off between minimizing the empirical error and controlling the model complexity. Controlling the model complexity is called *regularization*. In machine learning, we usually need to minimize the following *regularized empirical risk* [243]:

$$r(\boldsymbol{\theta}) + C * R_{emp}[f], \tag{B.7}$$

where $r(\boldsymbol{\theta})$ is the regularization term, θ is the model parameter, $R_{emp}[f]$ is the empirical error term, and C is a pre-specified parameter to balance the empirical error and model complexity.

Now, we give a typical example of learning a linear model by minimizing the regularized empirical risk. Suppose the loss function is squared loss $l(f,\boldsymbol{x},y) = (y - f(\boldsymbol{x}))^2$ and the regularization is l_2-norm $\frac{1}{2}\|\boldsymbol{w}\|_2^2$, then learning the linear model is the well-known *ridge*

regression [245]:

$$\frac{1}{2}\|\boldsymbol{w}\|_2^2 + C * \frac{1}{2}\|\boldsymbol{y} - (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{b})\|_2^2. \tag{B.8}$$

The *support vector machines* (SVMs) are well-known regularized sparse linear models. The hard-margin SVM was first proposed in 1979 [246], which was followed by the soft-margin SVM in 1995 [247]. The fundamental principle motivating it is that it implements the *structural risk minimization* (SRM) inductive principle [61] which states that the actual risk is upper bounded by the tradeoff between the empirical risk and model complexity. In the SRM theory, the structure $S_1 \subset S_2 \subset \cdots \subset S$ is constructed on the set of loss functions $S = \{Q(x, \alpha), \alpha \in \Lambda\}$. If the loss function is totally bounded, in order for the scheme to possess an ability of generalization, one needs to choose the function $Q(x, \alpha_n^k)$ from S_k that minimizes the bound $R(\alpha_n^k) \leq R_{emp}(\alpha_n^k) + \Phi(\frac{n}{h_k})$, where $R(\alpha_n^k)$ is the *actual* risk, $R_{emp}(\alpha_n^k)$ is the *empirical* risk, $\Phi(\frac{n}{h_k})$ is the confidence interval, n is the number of training samples, and h_k is the *Vapnik-Chervonenkis-dimension* (VC-dimension). For a fixed number of training samples, n, a small VC-dimension implies a small confidence interval.

We now consider how the SVM implements the SRM for linear models. SVM guarantees its generalization ability, because the SVM minimizes the empirical error, while maximizing the separating margin, which corresponds to the smallest VC-dimension. By mapping the original samples from the input space into a high-dimensional feature space using a basis function or kernel, one expects to have a smaller empirical error and a larger margin. Thus, SVM models can be expressed by the following general formula:

$$\min_{\boldsymbol{w},b} \sum_{i}^{n} l(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b, \boldsymbol{x}_i, y_i) + \frac{\lambda}{2}\|\boldsymbol{x}\|_2^2, \tag{B.9}$$

where $l(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b, \boldsymbol{x}_i, y_i)$ is a loss function, and λ controls the tradeoff between the approximation error and model complexity. For instance, the standard SVM applies the hinge loss (see Equation (B.2)).

In this appendix, we first review the main regularized (sparse) linear models using matrix notations, unlike element-wise notations in most of the existing reviews. The formulations using matrices and vectors are succinct and easy to understand. Second, two main feature selection techniques based on SVM are also reviewed in details. These techniques have been applied to gene selection problem in bioinformatics. Finally, we review the decomposition methods for two-class SVMs, and derive decomposition methods for one-class SVMs.

We shall use the following notations in the remaining of this appendix.

1. A matrix is represented by a boldface uppercase letter, for example $\boldsymbol{X}$. Its i-th column is represented by a boldface lowercase letter with a subscript, for example $\boldsymbol{x}_i$.

2. A (column) vector is denoted by a boldface lowercase letter, for example $\boldsymbol{y}$. We use MATLAB notations, and write $[y_1, y_2, \cdots, y_m]$ as a row vector, and $[y_1; y_2; \cdots, y_m]$ as a column vector.

3. A scalar is denoted by either a lower or uppercase letter, for example C and λ.

4. A constant column vector is represented by either a boldface symbol, for example $\boldsymbol{C}$ can be defined as $\boldsymbol{C} \in \{C\}^n$. Furthermore, in the bound constraints, the bounds are simply written as regular symbol without making them bold. For example, $\boldsymbol{\mu} \geq 0$ means all elements in vector $\boldsymbol{\mu}$ are greater than zero.

5. A constant matrix is represented by a boldface uppercase letter, for example $\boldsymbol{E}$ many contains constant 1.

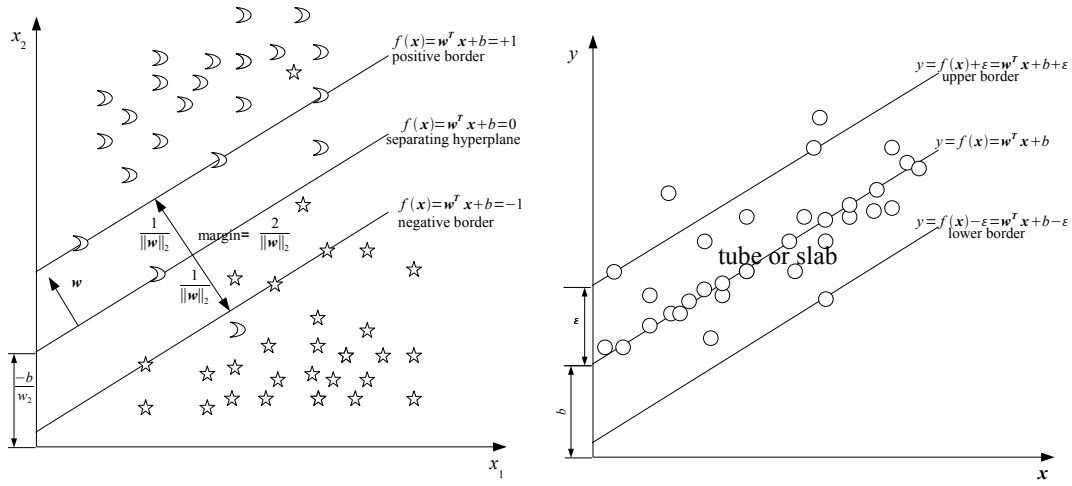
6. $\boldsymbol{X}_{m\times n}$ is a training set with m features and n samples.

7. For classification, these samples are from two groups: class -1 and class +1. The class labels of these n training samples are in the column vector $\boldsymbol{y}$.

8. For regression, the targets are real values accommodated in vector $\boldsymbol{y}$.

9. Matrix $\boldsymbol{S}_{m\times p}$ represents p unknown or test samples.

Furthermore, since the linear models have respective geometric interpretations, it is necessary to introduce some geometric concepts before discussing each model. In Figure B.1, we give schematic explanations for the related concepts in classification and regression. In classification, the hyperplane $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = 0$ is called the *separating hyperplane*. The area between $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = +1$ and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = -1$ is termed *margin*. Hyperplanes $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = +1$ and $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = -1$ are named *positive and negative borders* of the margin. In regression, The margin between $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b + \varepsilon$ and $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b - \varepsilon$ is called the *tube* or *slab*; and $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b + \varepsilon$ and $y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b - \varepsilon$ are the *upper and lower borders* of the slab, respectively.



(a) In classification. (b) In regression.

Figure B.1: The schematic explanation of some geometric conceptions. In (b), ε is specifically for the ε-insensitive loss.

2 A Review on Linear Models with Various Regularization and Loss Terms

In this section, we review the linear models with various regularization and loss functions. The regularized term uses either l_2 or l_1-norm. All loss functions mentioned in Equation (B.2) are discussed in details. We use vector and matrix notations instead of element-wise formulation. The benefit is that our formulations are very clear and easy to understand and implement in a programming language like MATLAB.

2.1 Hard-Margin l_2-Regularized Linear Model

With Bias b

The primal form of the hard-margin l_2-regularized linear model for classification and regression can be formulated as

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &\text{s.t. } \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b * \mathbf{1} = \boldsymbol{y}, \end{aligned} \tag{B.10}$$

where $\boldsymbol{X}$ is the training set, $\boldsymbol{y}$ is a column vector with targets, and $*$ is the operator of element-wise multiplication. Its corresponding Lagrange function is

$$L(\boldsymbol{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{\mu}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b * \mathbf{1}), \tag{B.11}$$

where $\boldsymbol{\mu}$ is a vector of Lagrange multipliers. From Equations (B.10) and (B.11), we can obtain the corresponding *Karush-Kuhn-Tucker* (KKT) conditions:

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} = 0 \Leftrightarrow \boldsymbol{w} = \boldsymbol{X}\boldsymbol{\mu} \\ \frac{\partial L}{\partial b} = -\boldsymbol{\mu}^{\mathrm{T}}\mathbf{1} = 0 \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b * \mathbf{1} = \boldsymbol{y} \end{cases} . \tag{B.12}$$

By substituting the primal variables with functions of Lagrange multipliers, the primal objective becomes the dual function:

$$g(\boldsymbol{\mu}) = -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu}. \tag{B.13}$$

Thus, we can obtain the dual form:

$$\begin{aligned} &\min_{\boldsymbol{\mu}} \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu} \\ &\text{s.t. } \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = 0. \end{aligned} \tag{B.14}$$

Obviously, this is an equality constrained *quadratic programming* (QP) problem that has a closed-form solution [26].

Once the optimal dual variable $\boldsymbol{\mu}$ is obtained, we can obtain the optimal primal variable using the KKT condition:

$$\boldsymbol{w} = \boldsymbol{X}\boldsymbol{\mu}. \tag{B.15}$$

The bias, b, can be obtained by the constraint:

$$\begin{aligned} b &= \mathrm{mean}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}) \\ &= \mathrm{mean}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu}). \end{aligned} \tag{B.16}$$

Thus, the linear model learned can be represented as:

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} + \mathrm{mean}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu}). \end{aligned} \tag{B.17}$$

We have the following remarks:

1. One of the strengthes of this model is that, the closed-form solution (Equations (B.15) and (B.16)) can be obtained. It makes the optimization very easy.

2. Also, only inner products of samples are involved in the optimization and prediction, therefore this model has a kernel version corresponding to the basis-expanded linear model.

3. However, all training samples are used in the optimization and decision making. The prediction phase becomes inefficient as the number of samples increases. The efficiency of online learning may also be affected by this weakness.

4. The hard margin assumption implies that the data are linearly separable in either input space or feature space. This is too strict for many applications.

5. We do not derive the optimization in sign-changed form like $\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} = \boldsymbol{y} * \boldsymbol{y}$ (where $\boldsymbol{z}_i = y_i * \boldsymbol{x}_i$) in the constraint, because this strategy only works when the target is non-zero. The sign-changing trick is used only to simplify the formulation in some cases where all targets are non-zeros.

Without Bias b

We will see that, for example in Sections 2.8 and 2.9, the bias b sometimes create difficulties during optimization. There are two tricks to circumvent it. The first one is to augment the feature vector by one, hence each sample becomes $[\boldsymbol{x}; 1]$, and the weight of the linear model becomes $[\boldsymbol{w}; b]$. The second one is to ignore the bias. It has been reported that the bias is not crucial in practice for high-dimensional data [248, 249]. In the following we present only the second trick, because the first trick can be derived by the same formula.

The primal form without considering the bias is

$$\begin{aligned} &\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &\text{s.t. } \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} = \boldsymbol{y}. \end{aligned} \tag{B.18}$$

Thus, the Lagrange function is

$$L(\boldsymbol{w}, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{\mu}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}). \tag{B.19}$$

The corresponding KKT conditions are

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} = 0 \Leftrightarrow \boldsymbol{w} = \boldsymbol{X}\boldsymbol{\mu} \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} = \boldsymbol{y} \end{cases}. \tag{B.20}$$

By removing the primal variables from the Lagrange function, one can obtain the dual function:

$$g(\boldsymbol{\mu}) = -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} + \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu}. \tag{B.21}$$

Therefore, the dual form is an unconstrained QP problem:

$$\min_{\boldsymbol{\mu}} \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu}. \tag{B.22}$$

From the first-order optimality condition, we have $\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{\mu} = \boldsymbol{y}$. Suppose the Hessian is positive definite, then it has the closed-form solution: $\boldsymbol{\mu} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{y}$. Therefore $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{y}$. The resulting linear function is therefore

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{x}. \tag{B.23}$$

We provide the following remarks to conclude this subsection:

1. The closed-form solution is very simple.
2. We can find that only inner products are involved. By replacing the inner products with kernel matrix, one have the kernel version.

2.2 Hard-Margin C-SVM: l_2-Regularized Linear Model

As mentioned above, the linear model in Equation (B.10) is too strict in some cases, thus the equality constraint is replaced with inequality in the hard-margin SVM [246]. In classification problem, the hard-margin SVM requires that the samples reside either on or outside

the border of the margin. It can be expressed as

$$\begin{cases} \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b \leq -1 & \text{if } y_i = -1; \\ \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b \geq +1 & \text{if } y_i = +1. \end{cases} \tag{B.24}$$

For simplicity, this constraint can be equivalently written as

$$y_i(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b) \leq 1. \tag{B.25}$$

For convenience of derivation, we use the equivalent matrix notation: $\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \leq \mathbf{1}$, where $\boldsymbol{Z}$ is sign-changed training set with its i-th column defined as the element-wise multiplication of the class label and the input vector of the i-th training sample, that is $\boldsymbol{z}_i = y_i * \boldsymbol{x}_i$. This trick does not apply for regression problem, because, for $y_i = 0$, $\boldsymbol{z}_i$ becomes $\mathbf{0}$.

The primal form can be written as

$$\begin{aligned} &\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 \\ &\text{s.t. } \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \mathbf{1}. \end{aligned} \tag{B.26}$$

Thus, the Lagrange function is

$$L(\boldsymbol{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{\mu}^{\mathrm{T}}(\mathbf{1} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}). \tag{B.27}$$

And the KKT conditions are

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}} = 0 \Leftrightarrow \boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} \\ \frac{\partial L}{\partial b} = -\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ \boldsymbol{\mu} * (\mathbf{1} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}) = \mathbf{0} \\ \boldsymbol{\mu} \geq 0 \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \mathbf{1} \end{cases}. \tag{B.28}$$

We thus have the dual function expressed as

$$\begin{aligned} g(\boldsymbol{\mu}) &= \inf_{\boldsymbol{w},b} L(\boldsymbol{w}, b, \boldsymbol{\mu}) \\ &= -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} + \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}. \end{aligned} \tag{B.29}$$

Therefore, the dual form of the optimization is

$$\begin{aligned} \min_{\boldsymbol{\mu}} & \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \\ \text{s.t. } & \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ & \boldsymbol{\mu} \geq 0. \end{aligned} \tag{B.30}$$

where $[\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}]_{ij} = l_i l_j(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)$. Note that though $[\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}]_{ij} = l_i l_j(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{x}_j)$, however, usually $K(\boldsymbol{Z},\boldsymbol{Z})_{ij} \neq l_i l_j K(\boldsymbol{x}_i,\boldsymbol{x}_j)$. Therefore, we need to compute kernel matrix using $\boldsymbol{X}$, but not $\boldsymbol{Z}$ directly.

After obtaining $\boldsymbol{\mu}$, we can compute the primal variables according to the KKT conditions. We have $\boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} = \boldsymbol{X}(\boldsymbol{\mu}*\boldsymbol{y})$. Let's define the set of indices $S = \{i|\mu_i > 0, i = 1:n\}$. Because if $\mu_i > 0$, then $1 = y_i(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b)$, that is $y_i = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b$. Therefore, $\boldsymbol{x}_i$ sits on the border (that is the hyperplane defined by either $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = -1$ or $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = 1$), and is called a *support vector*. By using the support vectors only, we have the solution:

$$\boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} = \boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S). \tag{B.31}$$

The bias can be computed using the points on the border: $b = y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w}$ where $i \in S$. For stable result, we use all support vectors, and therefore

$$b = \frac{\boldsymbol{y}_S - \boldsymbol{X}_S^{\mathrm{T}}\boldsymbol{w}}{|S|} = \frac{\boldsymbol{y}_S - \boldsymbol{X}_S^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)}{|S|}. \tag{B.32}$$

The resulting linear function can hence be represented by a function related to the support vectors:

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S) + \frac{\boldsymbol{y}_S - \boldsymbol{X}_S^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)}{|S|}. \end{aligned} \tag{B.33}$$

Remarks:

1. Only the support vectors are needed in the prediction, therefore the hard-margin SVM can be applied for sample selection.
2. Inner products are involved in the optimization and prediction, therefore one can kernelize it.
3. The margin is still hard, as no samples are allowed within the margin.

2.3 Soft-Margin C-SVM: l_2-Regularized Hinge-Loss Linear Model

The hard-margin SVM works only for linearly separable case. The soft-margin SVM [247] allows the relaxation on the constraints, therefore it is applicable on non-linearly separable data. Comparing with the hard-margin SVM, soft-margin SVM uses the following constraint

$$\begin{cases} \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b \leq -1 + \xi_i & \text{if } y_i = -1; \\ \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b \geq +1 - \xi_i & \text{if } y_i = +1, \end{cases} \tag{B.34}$$

where the relaxation variable $\xi_i \geq 0$. For simplicity, this can be rewritten into the equivalent form: $\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi}$. The soft-margin SVM wants to maximize the margin while minimizing the relaxation. Therefore, the optimization task of soft-margin SVM can be expressed in the following equation:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ &\text{s.t. } \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi} \\ &\qquad \boldsymbol{\xi} \geq 0. \end{aligned} \tag{B.35}$$

The corresponding Lagrange function is

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\eta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \boldsymbol{\mu}^{\mathrm{T}}(\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}) - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi}. \tag{B.36}$$

And the KKT conditions are

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}} \Leftrightarrow \boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} \\ \frac{\partial L}{\partial b} = -\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \Leftrightarrow \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ \frac{\partial L}{\partial \boldsymbol{\xi}} = \boldsymbol{C}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{\eta}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}} \Leftrightarrow \boldsymbol{C} - \boldsymbol{\mu} - \boldsymbol{\eta} = \mathbf{0} \\ \boldsymbol{\mu} * (\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}) = \mathbf{0} \\ \boldsymbol{\eta} * \boldsymbol{\xi} = \mathbf{0} \\ \mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y} \leq 0 \\ \boldsymbol{\xi} \geq 0 \\ \boldsymbol{\mu} \geq 0 \\ \boldsymbol{\eta} \geq 0 \end{cases}. \tag{B.37}$$

One can obtain the dual function:

$$g(\boldsymbol{\mu}, \boldsymbol{\eta}) = \inf_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\eta}) = -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} + \boldsymbol{\mu}^{\mathrm{T}}\mathbf{1}. \tag{B.38}$$

Therefore, the dual form is formulated as

$$\begin{aligned} \min_{\boldsymbol{\mu}} & g(\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} - \boldsymbol{\mu}^{\mathrm{T}}\mathbf{1} \\ \text{s.t. } & \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ & \boldsymbol{c} = \boldsymbol{\mu} + \boldsymbol{\eta} \\ & \boldsymbol{\mu} \geq 0 \\ & \boldsymbol{\eta} \geq 0. \end{aligned} \tag{B.39}$$

This is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\mu}} & g(\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \\ \text{s.t. } & \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ & 0 \leq \boldsymbol{\mu} \leq C \end{aligned} \tag{B.40}$$

Similar to the hard-margin SVM, the normal vector is a non-negative linear combination of the training samples, that is $\boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} = \boldsymbol{X}(\boldsymbol{\mu} * \boldsymbol{y}) = \boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)$, where S is the set of indices of non-zero multipliers: $S = \{i|\mu_i > 0, i = 1, \cdots, n\}$. The training samples corresponding to S are called *support vectors* (SVs), as they are either on the correct margin border or at the wrong side. In order to compute the bias b, we need to find some points on their corresponding boundary, denoted by $\boldsymbol{X}_B$, where $B = \{i|0 < \mu_i < C, i = 1, \cdots, n\}$. We have $b = \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{w}}{|B|} = \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)}{|B|}$.

After obtaining the optimal $\boldsymbol{w}$ and b, the linear function used by the decision function can be computed as below:

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S) + \frac{\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_S * \boldsymbol{y}_S)}{|B|}. \tag{B.41}$$

According to the KKT conditions, we can obtain the following important geometric interpretations associated with the optimal multipliers.

1. If $\mu_i > 0$, then training point $\boldsymbol{x}_i$ resides either on or at the side, where $-y_i\boldsymbol{w}$ is pointing, of its margin border (If $y_i f(\boldsymbol{x}_i) < 1$, we say $\boldsymbol{x}_i$ is outside the border corresponding

to y_i; otherwise, inside.). However, the reverse is not true (we shall next point out that, if $\boldsymbol{x}_i$ is on its correct margin border, then it is possible that $\mu_i = 0$). We call points with nonzero multipliers *support vectors*. This judgement is proven as follows. If $\mu_i > 0$, from complementarity condition we then have $1 - \xi_i - \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} - by_i = 0$. From $\eta_i = C - \mu_i$, we have $0 \leq \eta_i < C$. If $0 < \mu_i < C$, then $\eta_i > 0$, then ξ_i=0, therefore $1 - \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} - by_i = 0$, that is $\boldsymbol{x}_i$ is on the correct margin border. If $\mu_i = C$, then $\eta_i = 0$, then $\xi_i \geq 0$, then $1 - \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} - by_i \geq 0$, that is $\boldsymbol{x}_i$ is either on or at the side, where $-y_i\boldsymbol{w}$ is pointing, of its correct margin border.

2. If $0 < \mu_i < C$, then $\boldsymbol{x}_i$ is on the margin border. It can be seen from the proof of the first point. However, the reverse of this proposition is not always true. We can state that, if $\boldsymbol{x}_i$ is on the correct margin border, then $0 \leq \mu_i \leq C$. We can briefly prove it in the following. The condition, that $\boldsymbol{x}_i$ is on the correct margin border, is equivalent to $1 - \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} - by_i = 0$. This implies that $\xi_i = 0$. From its complementarity condition, we have $\eta_i \geq 0$. From condition $\mu_i = C - \eta_i$, we thus have $0 \leq \mu_i \leq C$. Please note that, we do not treat the point with zero multiplier as a support vector.

3. If $\boldsymbol{x}_i$ is at the side, where $-y_i\boldsymbol{w}$ is pointing, of the corresponding margin border, then $\mu_i = C$. However, the reverse is not always true. If $\mu_i = C$, then $\boldsymbol{x}_i$ is either on or at the side, where $-y_i\boldsymbol{w}$ is pointing, of its correct margin border.

2.4 ν-SVM: Another l_2-Regularized Hinge-Loss Linear Model

Suppose that $\boldsymbol{x}$ is a point located on its correct border of the margin. Its corresponding value of $f(\boldsymbol{x})$ can be written as $y\rho$ (where $\rho \geq 0$ and $y \in \{-1, +1\}$ is the class label of $\boldsymbol{x}$), then the margin between positive and negative margin borders becomes $\frac{2\rho}{\|\boldsymbol{w}\|_2}$. In C-SVM, ρ is fixed by 1, and the margin in C-SVM is controlled by parameter C. Alternatively, it can be controlled by adjusting the coefficient of ρ as in ν-SVM [157]. The primal form of ν-SVM can be formulated as below:

$$\begin{aligned} \min_{\boldsymbol{w},b,\rho,\boldsymbol{\xi}} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 - C_0\nu\rho + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t.} \ & \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \rho\mathbf{1} - \boldsymbol{\xi} \\ & \boldsymbol{\xi} \geq 0 \\ & \rho \geq 0, \end{aligned} \tag{B.42}$$

where C_0 and ν are pre-specified parameters, and $\boldsymbol{C}$ is a column vector that takes instant value $C = \frac{C_0}{n}$ (we shall show that C_0 can be simply set to 1 later). $\boldsymbol{Z}$ is the sign-changed training set as in C-SVM. From the above equation, the corresponding Lagrange function is

$$L(\boldsymbol{w}, b, \rho, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\eta}, \delta) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 - C_0\nu\rho + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \boldsymbol{\mu}^{\mathrm{T}}(\rho\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}) - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi} - \delta\rho. \tag{B.43}$$

The corresponding KKT conditions are

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}} \Leftrightarrow \boldsymbol{w} = \boldsymbol{Z}\boldsymbol{\mu} \\ \frac{\partial L}{\partial b} = -\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \Leftrightarrow \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{y} = 0 \\ \frac{\partial L}{\partial \boldsymbol{\xi}} = \boldsymbol{C}^{\mathrm{T}} - \boldsymbol{\mu}^{\mathrm{T}} - \boldsymbol{\eta}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}} \Leftrightarrow \boldsymbol{C} - \boldsymbol{\mu} - \boldsymbol{\eta} = \mathbf{0} \\ \frac{\partial L}{\partial \rho} = \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} - C_0\nu - \delta = 0 \\ \boldsymbol{\mu} * (\rho\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}) = \mathbf{0} \\ \boldsymbol{\eta} * \boldsymbol{\xi} = \mathbf{0} \\ \delta * \rho = 0 \\ \boldsymbol{\mu} \geq 0 \\ \boldsymbol{\eta} \geq 0 \\ \rho\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y} \leq 0 \\ \boldsymbol{\xi} \geq 0 \\ \boldsymbol{\rho} \geq 0 \end{cases}. \tag{B.44}$$

Thus, the dual function is

$$g(\boldsymbol{\mu}, \boldsymbol{\eta}, \delta) = \inf_{\boldsymbol{w}, b, \boldsymbol{\xi}} L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\eta}) = -\frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu}. \tag{B.45}$$

And the dual form is

$$\begin{aligned} \min_{\boldsymbol{\mu}} & g(\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z}\boldsymbol{\mu} \\ \text{s.t. } & \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu} = 0 \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \geq C_0\nu \\ & 0 \leq \boldsymbol{\mu} \leq C. \end{aligned} \tag{B.46}$$

Once we obtain the solution to the dual form, we can obtain the optimal solution to $\boldsymbol{w}$ of the primal form according to the KKT condition: $\boldsymbol{w} = \boldsymbol{Z}_S\boldsymbol{\mu}_S$ where S is the set of indices of nonzero multipliers. However, we can not use the same way as C-SVM to compute the bias b, because the points on the margin satisfy $y_i(\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b) = \rho$ where ρ is also unknown. We can use the following trick, proposed in [157], to solve it. First, we find s positive points, denoted by $\boldsymbol{X}_+$ (which are on the positive border) and s negative points, denoted by $\boldsymbol{X}_-$ (which are on the negative border). Then we have

$$\boldsymbol{X}_+^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} = \boldsymbol{\rho} = -\boldsymbol{X}_-^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1}. \tag{B.47}$$

Therefore we can obtain the optimal bias:

$$b = -\frac{1}{2}\mathrm{mean}((\boldsymbol{X}_+ + \boldsymbol{X}_-)^{\mathrm{T}}\boldsymbol{w}) = -\frac{1}{2}\mathrm{mean}((\boldsymbol{X}_+ + \boldsymbol{X}_-)^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * \boldsymbol{\mu}_S)). \tag{B.48}$$

The linear function in the decision function is

$$f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * \boldsymbol{\mu}_S) - \frac{1}{2}\mathrm{mean}((\boldsymbol{X}_+ + \boldsymbol{X}_-)^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * \boldsymbol{\mu}_S)). \tag{B.49}$$

From the KKT conditions, the ν-SVM has the following properties:

1. From the dual form, one can see that the objective is homogeneous, scaling the variable $\boldsymbol{\mu}$ would not change the decision function. Therefore, one can simply set $C_0 = 1$. The last two constraints are $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \geq \nu$ and $\mathbf{0} \leq \boldsymbol{\mu} \leq \frac{\mathbf{1}}{\boldsymbol{n}}$ then.

2. An error is defined as the training sample that resides on the wrong side of its margin border. If $\rho > 0$, then ν is an upper bound on the fraction of errors. That is $\nu \geq \frac{n_e}{n}$, where n_e is the number of errors. It is briefly proven in the following. Because $\rho > 0$ and $\delta\rho = 0$, therefore $\delta = 0$. Because $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} - \nu - \delta = 0$, therefore $\nu = \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}$. Because the upper-bound is $\frac{1}{n}$, therefore $\nu = \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \geq \frac{n_C}{n}$, where n_C is the number of upper-bounded multipliers. Because $n_C \geq n_e$, we therefore have $\nu = \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \geq \frac{n_C}{n} \geq \frac{n_e}{n}$.

3. The support vectors are defined as the training samples corresponding to the non-zero multipliers which correspond to the active constraints $\rho\mathbf{1} - \boldsymbol{\xi}_S - \boldsymbol{Z}_S^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{y}_S = 0$. That is $\boldsymbol{Z}_S^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y}_S \leq \rho\mathbf{1}$. Therefore, the support vectors are subset of training samples either on the correct margin border or at its wrong side. If $\rho > 0$, then ν is a lower bound on the fraction of support vectors. That is $\nu \leq \frac{n_S}{n}$.

4. The range of ν is $(0, 1)$, while the range of C in C-SVM is $(0, +\infty)$. Therefore, it is more convenient in practice to use ν-SVM than C-SVM for model selection.

5. The conclusions of C-SVM regarding the relations between multiplier and point position apply to ν-SVM as well.

2.5 l_2-Regularized l_1-Loss Linear Model

The l_2-regularized l_1-loss linear model can be applied to both classification and regression. We only derive it for classification problem below. Its formulation for regression is similar to the ε-SVR that will be derived in Section 2.6. Its primal form can be expressed as

$$\begin{aligned} \min_{\boldsymbol{w},b} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t. } & \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi} \\ & \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} \leq \mathbf{1} + \boldsymbol{\xi} \\ & \boldsymbol{\xi} \geq 0, \end{aligned} \tag{B.50}$$

where the constraint is equivalent to $-\boldsymbol{\xi} \leq \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} - \mathbf{1} \leq \boldsymbol{\xi}$, that is $|\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i - 1| \leq \xi_i$ for $\forall i = 1, \cdots, n$. Its Lagrange function is

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\eta}) = & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \boldsymbol{\mu}_1^{\mathrm{T}}(\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b * \boldsymbol{y}) \\ & + \boldsymbol{\mu}_2^{\mathrm{T}}(-\mathbf{1} - \boldsymbol{\xi} + \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y}) - \boldsymbol{\eta}\boldsymbol{\xi}. \end{aligned} \tag{B.51}$$

Thus, the corresponding KKT conditions are

$$\begin{cases} \frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}} - (\boldsymbol{\mu}_1 - \boldsymbol{\mu_2})^{\mathrm{T}} \boldsymbol{Z}^{\mathrm{T}} = 0 \Leftrightarrow \boldsymbol{w} = \boldsymbol{Z}(\boldsymbol{\mu}_1 - \boldsymbol{\mu_2}) \\ \frac{\partial L}{\partial b} = \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu_2}) = 0 \\ \frac{\partial L}{\partial \boldsymbol{\xi}} = \boldsymbol{C}^{\mathrm{T}} - \boldsymbol{\mu}_1^{\mathrm{T}} - \boldsymbol{\mu}_2^{\mathrm{T}} - \boldsymbol{\eta}^{\mathrm{T}} = 0 \Leftrightarrow \boldsymbol{\mu}_1 + \boldsymbol{\mu_2} = \boldsymbol{C} - \boldsymbol{\eta} \\ \boldsymbol{\mu}_1 * (\mathbf{1} - \boldsymbol{\xi} - \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} - b * \boldsymbol{y}) = \mathbf{0} \\ \boldsymbol{\mu}_2 * (-\mathbf{1} - \boldsymbol{\xi} + \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y}) = \mathbf{0} \\ \boldsymbol{\eta} * \boldsymbol{\xi} = \mathbf{0} \\ \boldsymbol{\mu}_1 \geq 0 \\ \boldsymbol{\mu}_2 \geq 0 \\ \boldsymbol{\eta} \geq 0 \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi} \\ \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b * \boldsymbol{y} \leq \mathbf{1} + \boldsymbol{\xi} \\ \boldsymbol{\xi} \geq 0. \end{cases} . \tag{B.52}$$

Eliminating the primal variables in the Lagrange function using the KKT conditions, we can obtain the dual function as below:

$$g(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) = -\frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}} \boldsymbol{Z}^{\mathrm{T}} \boldsymbol{Z}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) + \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2). \tag{B.53}$$

Thus, the dual form is

$$\begin{aligned} \min_{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2} & \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}} \boldsymbol{Z}^{\mathrm{T}} \boldsymbol{Z}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ \text{s.t. } & \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = 0 \\ & \boldsymbol{\mu}_1 + \boldsymbol{\mu}_2 \leq C \\ & \boldsymbol{\mu}_1, \boldsymbol{\mu}_2 \geq 0. \end{aligned} \tag{B.54}$$

We can see that, this is also a constrained QP problem. According to our discussion at end of this subsection, we can define $S = \{\forall i = 1, \cdots, n, \mu_{1i} - \mu_{2i} \neq 0\}$ to be the set of indices of the support vectors, and $B = \{\forall i = 1, \cdots, n, \mu_{1i} + \mu_{2i} < C\}$ to be the set of indices of the points residing on the corresponding margin borders. Then, the normal vector can be

expressed as

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{X}(\boldsymbol{y} * (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)) \\ &= \boldsymbol{X}_S(\boldsymbol{y}_S * (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S). \end{aligned} \tag{B.55}$$

And the bias b can be computed as

$$\begin{aligned} b &= \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{w}) \\ &= \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S). \end{aligned} \tag{B.56}$$

Thus, this linear model can be written as

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S) + \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{y}_S * (\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S). \end{aligned} \tag{B.57}$$

We can see that this model requires only inner products of samples, rather than the original samples. Therefore, it can be extended to kernel version by replacing these inner products by kernel matrices.

The followings are the geometric interpretation of the Lagrange multipliers of this problem.

1. If $\mu_{1i}, \mu_{2i} > 0$ and $\mu_{1i} + \mu_{2i} < C$, then $\boldsymbol{x}_i$ is on its corresponding margin border $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b = y_i$. We can prove it in the following. From $\mu_{1i}, \mu_{2i} > 0$, we have

$$\begin{cases} \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i = 1 - \xi_i \\ \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i = 1 + \xi_i \end{cases}. \tag{B.58}$$

   From $\mu_{1i} + \mu_{2i} < C$ and $\mu_{1i} + \mu_{2i} = C - \eta_i$, we have $\eta_i > 0$. From $\eta_i\xi_i = 0$, we then have $\xi_i = 0$. Thus $\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i = 1$, that is $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = y_i$. So $\boldsymbol{x}_i$ is on its corresponding margin border.

2. If $\mu_{1i} = \mu_{2i} = 0$, thus from the KKT condition $\mu_{1i} + \mu_{2i} = C - \eta_i$, thus $\eta_i = C$. From $\eta_i\xi_i = 0$, we have $\xi_i = 0$. Thus, $\boldsymbol{x}_i$ resides on its corresponding margin border.

3. If $\mu_{1i} = 0$ and $\mu_{2i} > 0$, then $\boldsymbol{x}_i$ is either on or at the side, where $y_i\boldsymbol{w}$ is pointing, of

its corresponding margin border. From $\mu_{1i} = 0$ and $\mu_{2i} > 0$, we have

$$\begin{cases} \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i \geq 1 - \xi_i \\ \boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i = 1 + \xi_i \end{cases}. \tag{B.59}$$

Then $\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w} + by_i = 1 + \xi_i$, that is $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = y_i + y_i\xi_i$. If $y_i = 1$, then $\boldsymbol{x}_i$ is either on or at the side, where $\boldsymbol{w}$ is pointing, of $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = 1$. Similarly, if $y_i = -1$, then $\boldsymbol{x}_i$ is either on or at the side, where $-\boldsymbol{w}$ is pointing, of $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = -1$.

4. If $\mu_{1i} = 0$ and $0 < \mu_{2i} < C$, then $\eta_i > 0$, thus $\xi_i = 0$. Therefore, we are sure that $\boldsymbol{x}_i$ is located on its corresponding margin border.

5. Similarly, if $\mu_{1i} > 0$ and $\mu_{2i} = 0$, then $\boldsymbol{x}_i$ resides either on or at the side, where $-y_i\boldsymbol{w}$ is pointing, of its corresponding margin border. If $0 < \mu_{1i} < C$ and $\mu_{2i} = 0$, then $\boldsymbol{x}_i$ is located on its corresponding margin border.

6. From the above discussion, we can also know that if $\mu_{1i} + \mu_{2i} < C$, then $\boldsymbol{x}_i$ is located on its corresponding margin border. If $\boldsymbol{x}_i$ is not on its corresponding margin border, then $\mu_{1i} + \mu_{2i} = C$.

7. From the KKT condition $\boldsymbol{w} = \boldsymbol{Z}(\boldsymbol{\mu}_1 - \boldsymbol{\mu_2})$, it can be seen that the optimal normal vector is a linear combination of the training samples. We call the points satisfying $\mu_{1i} - \mu_{2i} \neq 0$ as *support vectors*. From the above discussion, we find that the support vectors can locate anywhere of the space.

8. Unlike the linear model using hinge-loss, only the points that reside exactly on their corresponding hyperplane ($\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b = y_i$) are not penalized when using l_1-loss.

9. We will show, below, that the l_1-loss is a special case of the ε-insensitive loss for regression problems.

2.6 ε-SVR: l_2-Regularized ε-Insensitive-Loss Linear Model

From Equation (B.2), we can find that the l_1-loss is a specific form of ε-insensitive loss by setting $\varepsilon = 0$. Using ε-insensitive loss, one can obtain the sparse linear models for regression, coined *ε-support vector regression* (ε-SVR) [250]. Suppose the loss function is the ε-insensitive loss function

$$l(f, \boldsymbol{x}, y) = |y - f(\boldsymbol{x})|_\varepsilon = \max\{0, |y - f(\boldsymbol{x})| - \varepsilon\}, \tag{B.60}$$

then the l_2-regularized linear model is formulated as

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \frac{C_0}{n}\sum_{i=1}^{n}\max\{0, |y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b| - \varepsilon\}. \tag{B.61}$$

We treat the constant $\frac{C_0}{n}$ by one symbol $\frac{C_0}{n} = C$, then we have

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\max\{0, |y_i - \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_i - b| - \varepsilon\}. \tag{B.62}$$

The loss term is non-smooth (indifferentiable). We can transform the above equation to the equivalent form which is smooth:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}_1,\boldsymbol{\xi}_2} \ & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2) \\ \text{s. t. } & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b - \boldsymbol{y} \leq \varepsilon + \boldsymbol{\xi}_1 \\ & \boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b \leq \varepsilon + \boldsymbol{\xi}_2 \\ & \boldsymbol{\xi}_1 \geq 0 \\ & \boldsymbol{\xi}_2 \geq 0. \end{aligned} \tag{B.63}$$

Its Lagrange function is

$$\begin{aligned} L(\boldsymbol{w}, b, \boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\eta}_1, \boldsymbol{\eta}_2) = & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2) \\ & + \boldsymbol{\mu}_1^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{1} - \boldsymbol{y} - \varepsilon\boldsymbol{1} - \boldsymbol{\xi}_1) \\ & + \boldsymbol{\mu}_2^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{1} - \varepsilon\boldsymbol{1} - \boldsymbol{\xi}_2) \\ & - \boldsymbol{\eta}_1^{\mathrm{T}}\boldsymbol{\xi}_1 - \boldsymbol{\eta}_2^{\mathrm{T}}\boldsymbol{\xi}_2. \end{aligned} \tag{B.64}$$

The corresponding KKT conditions are

$$\begin{cases} \boldsymbol{w} = \boldsymbol{X}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = 0 \\ \boldsymbol{C} - \boldsymbol{\mu}_1 - \boldsymbol{\eta}_1 = \mathbf{0} \\ \boldsymbol{C} - \boldsymbol{\mu}_2 - \boldsymbol{\eta}_2 = \mathbf{0} \\ \boldsymbol{\mu}_1 * (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} - \boldsymbol{y} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_1) = \mathbf{0} \\ \boldsymbol{\mu}_2 * (\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_2) = \mathbf{0} \\ \boldsymbol{\eta}_1 * \boldsymbol{\xi}_1 = \mathbf{0} \\ \boldsymbol{\eta}_2 * \boldsymbol{\xi}_2 = \mathbf{0} \\ \boldsymbol{\mu}_1 \geq 0 \\ \boldsymbol{\mu}_2 \geq 0 \\ \boldsymbol{\eta}_1 \geq 0 \\ \boldsymbol{\eta}_2 \geq 0 \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} - \boldsymbol{y} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_1 \leq 0 \\ \boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_2 \leq 0 \\ \boldsymbol{\xi}_1 \geq 0 \\ \boldsymbol{\xi}_2 \geq 0 \end{cases} . \tag{B.65}$$

Therefore, the dual form is

$$\begin{aligned} \min_{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2} & \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \boldsymbol{\varepsilon}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) \\ \text{s.t.} \ & \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = 0 \\ & 0 \leq \boldsymbol{\mu}_1 \leq C \\ & 0 \leq \boldsymbol{\mu}_2 \leq C. \end{aligned} \tag{B.66}$$

After obtaining the dual solution $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$, the normal vector $\boldsymbol{w}$ can be computed using the KKT condition: $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$. From the remarks at the end of this subsection, we know that, if $0 < \mu_{1i} < C$ or $0 < \mu_{2i} < C$, then $\boldsymbol{x}_i$ resides on the lower or upper border. If $\mu_{1i} > 0$ or $\mu_{2i} > 0$, then $\boldsymbol{x}_i$ reside either on the border or outside the slab, and is a support vector. Define B as the set of indices that satisfy $0 < \mu_{1i} < C$ or $0 < \mu_{2i} < C$: $B = \{i | \forall i = 1, \cdots, n, 0 < \mu_{1i} < C \vee 0 < \mu_{2i} < C\}$. Define S as the set of indices that satisfy

$\mu_{1i} > 0$ or $\mu_{2i} > 0$: $S = \{i | \forall i = 1, \cdots, n, \mu_{1i} > 0 \vee \mu_{2i} > 0\}$. Then, the normal vector can be expressed as

$$\begin{aligned} \boldsymbol{w} &= \boldsymbol{X}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ &= \boldsymbol{X}_S(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S. \end{aligned} \tag{B.67}$$

And the bias b can be computed as

$$\begin{aligned} b &= \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{w}) \\ &= \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S). \end{aligned} \tag{B.68}$$

Thus, this linear model can be written as

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{w} + b \\ &= \boldsymbol{x}^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S + \text{mean}(\boldsymbol{y}_B - \boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)_S). \end{aligned} \tag{B.69}$$

We can see that the ε-SVR model requires inner products of samples. Therefore, it can be extended to kernel version by replacing these inner products by kernel matrices.

We have the following remarks on ε-SVR:

1. The second term of the primal objective can be rewritten as $C\|\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2\|_1$. Therefore we can see that the slack variables should be sparse.

2. If $\mu_{1i} > 0$, then $\boldsymbol{x}_i$ is either on or underneath the lower border. This is because, from $\mu_{1i} > 0$, we know $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b - y_i = \varepsilon + \xi_{1i}$, that is $y_i = f(\boldsymbol{x}_i) - \varepsilon - \xi_{1i} \leq f(\boldsymbol{x}_i) - \varepsilon$. Recall that, in the context of regression, the margin is called *tube* or *slab*, and the borders of the slab are called upper and lower *borders*.

3. Similarly, if $\mu_{2i} > 0$, then $\boldsymbol{x}_i$ is either on or above the upper border.

4. If $\mu_{1i} = 0$, then $\boldsymbol{x}_i$ is either on or above the lower border. From $\mu_{1i} = 0$, we have $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b - y_i \leq \varepsilon + \xi_{1i}$. From $C - \mu_{1i} - \eta_{1i} = 0$, we have $\eta_{1i} = C$, we then have $\xi_{1i} = 0$. Therefore $y_i \geq f(\boldsymbol{x}_i) - \varepsilon$. Thus, $\boldsymbol{x}_i$ is on or above the lower border.

5. Similarly, if $\mu_{2i} = 0$, then $\boldsymbol{x}_i$ is either on or under the upper border.

6. If $\mu_{1i} = \mu_{2i} = 0$, then $\boldsymbol{x}_i$ is either on the borders or inside the slab.

7. If $\mu_{1i} > 0$ and $\mu_{2i} = 0$, then $\boldsymbol{x}_i$ is either on or below the lower border.

8. If $\mu_{1i} = 0$ and $\mu_{2i} > 0$, then $\boldsymbol{x}_i$ is either on or above the upper border.

9. For all i, $\mu_{1i}\mu_{2i} = 0$ must be satisfied. That is, either μ_{1i} or μ_{2i} must be zero. We can conduct proof by contradiction. Suppose $\mu_{1i} > 0$ and $\mu_{2i} > 0$, then $\boldsymbol{x}_i$ is on-or-under the lower border and on-or-above the upper border. This is impossible. It can also be proven as follows. If $\boldsymbol{x}_i$ is outside the upper border, then $\xi_{2i} = y_i - \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} - b - \varepsilon > 0$, then $\eta_{2i} = 0$, then $\mu_{2i} = C$. Meanwhile, $\boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b - y_i - \varepsilon - \xi_{1i} < 0$ ($\xi_{1i} = 0$ due to the objective), then $\mu_{1i} = 0$ (because of the complementary condition). If $\boldsymbol{x}_i$ is on the upper border, then $\xi_{2i} = 0$, then $\eta_{2i} \geq 0$, then $0 < \mu_{2i} \leq C$, and $\mu_{1i} = 0$. If $\boldsymbol{x}_i$ is inside the slab, then $\mu_{1i} = \mu_{2i} = 0$. If $\boldsymbol{x}_i$ is on the lower border, then $\mu_{2i} = 0$ and $0 < \mu_{1i} \leq C$. If $\boldsymbol{x}_i$ is below the lower border, then $\mu_{2i} = 0$ and $\mu_{1i} = C$.

10. Because of $\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1)$, the training samples corresponding to nonzero $\boldsymbol{\mu}_1$ or $\boldsymbol{\mu}_2$ are called *support vectors*. If a sample $\boldsymbol{x}_i$ is a support vector, then from the complementary condition, it satisfies either $y_i = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b - \varepsilon - \xi_{1i}$ or $y_i = \boldsymbol{x}_i^{\mathrm{T}}\boldsymbol{w} + b + \varepsilon + \xi_{2i}$. Therefore, support vectors consist of points that are either on or outside the tube.

11. If a point is outside a slab, then the corresponding multiplier equals C. If it is on a border, then the corresponding multiplier is in the range $(0, C]$. If it is inside the slab, then the corresponding multiplier is zero. The KKT conditions $\boldsymbol{C} - \boldsymbol{\mu}_1 - \boldsymbol{\eta}_1 = 0$ and $\boldsymbol{C} - \boldsymbol{\mu}_2 - \boldsymbol{\eta}_2 = 0$ are important to tell us the relation between the Lagrange multipliers and the slack variables. We can obtain the following relations. $\mu_{1(2)i} = C \Leftrightarrow \eta_{1(2)i} = 0 \Leftrightarrow \xi_{1(2)i} \geq 0$. From $\xi_{1(2)i} > 0 \Rightarrow \eta_{1(2)i} = 0 \Rightarrow \mu_{1(2)i} = C$. Therefore we can state that a nonzero slack variable implies that the corresponding multiplier μ takes the upper bound. However, the number of upper-bounded multipliers is greater than or equal to the number of errors, where an error is defined as a training sample that resides outside the tube.

2.7 ν-SVR: Another l_2-Regularized ε-Insensitive-Loss Linear Model

The radius of the tube is ε which is pre-specified *a priori* in ε-SVR. However, its proper value is unknown. ν-SVR [251] can minimize ε and therefore has the advantage of controlling error. It turns out ν-SVR with objective: $\frac{1}{2}\|\boldsymbol{w}\|_2^2 + C_0(\nu\varepsilon + (\frac{\boldsymbol{1}}{\boldsymbol{n}})^{\mathrm{T}}(\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2))$.

We let $\boldsymbol{C} = \{\frac{C_0}{n}\}^n$. The ν-SVR is formulated as below:

$$\begin{aligned} \min_{\boldsymbol{w},b,\varepsilon,\boldsymbol{\xi}_1,\boldsymbol{\xi}_2} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C_0\nu\varepsilon + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2) \\ \text{s. t. } & \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b - \boldsymbol{y} \leq \varepsilon + \boldsymbol{\xi}_1 \\ & \boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b \leq \varepsilon + \boldsymbol{\xi}_2 \\ & \boldsymbol{\xi}_1 \geq 0 \\ & \boldsymbol{\xi}_2 \geq 0 \\ & \varepsilon \geq 0. \end{aligned} \tag{B.70}$$

By introducing Lagrange multipliers, we can obtain the corresponding Lagrange function:

$$\begin{aligned} L(\boldsymbol{w}, b, \varepsilon, \boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\eta}_1, \boldsymbol{\eta}_2, \beta) = & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C_0\nu\varepsilon + \boldsymbol{C}^{\mathrm{T}}(\boldsymbol{\xi}_1 + \boldsymbol{\xi}_2) \\ & + \boldsymbol{\mu}_1^{\mathrm{T}}(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{1} - \boldsymbol{y} - \varepsilon\boldsymbol{1} - \boldsymbol{\xi}_1) \\ & + \boldsymbol{\mu}_2^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\boldsymbol{1} - \varepsilon\boldsymbol{1} - \boldsymbol{\xi}_2) \\ & - \boldsymbol{\eta}_1^{\mathrm{T}}\boldsymbol{\xi}_1 - \boldsymbol{\eta}_2^{\mathrm{T}}\boldsymbol{\xi}_2 \\ & - \beta\varepsilon. \end{aligned} \tag{B.71}$$

Then, the KKT conditions can be obtained:

$$\begin{cases} \boldsymbol{w} = \boldsymbol{X}(\boldsymbol{\mu}_2 - \boldsymbol{\mu}_1) \\ \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = 0 \\ C_0\nu - \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \beta = 0 \\ \boldsymbol{C} - \boldsymbol{\mu}_1 - \boldsymbol{\eta}_1 = \mathbf{0} \\ \boldsymbol{C} - \boldsymbol{\mu}_2 - \boldsymbol{\eta}_2 = \mathbf{0} \\ \boldsymbol{\mu}_1 * (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} - \boldsymbol{y} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_1) = \mathbf{0} \\ \boldsymbol{\mu}_2 * (\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_2) = \mathbf{0} \\ \boldsymbol{\eta}_1 * \boldsymbol{\xi}_1 = \mathbf{0} \\ \boldsymbol{\eta}_2 * \boldsymbol{\xi}_2 = \mathbf{0} \\ \beta\varepsilon = 0 \\ \boldsymbol{\mu}_1 \geq 0 \\ \boldsymbol{\mu}_2 \geq 0 \\ \boldsymbol{\eta}_1 \geq 0 \\ \boldsymbol{\eta}_2 \geq 0 \\ \beta \geq 0 \\ \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1} - \boldsymbol{y} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_1 \leq 0 \\ \boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} - \varepsilon\mathbf{1} - \boldsymbol{\xi}_2 \leq 0 \\ \boldsymbol{\xi}_1 \geq 0 \\ \boldsymbol{\xi}_2 \geq 0 \\ \boldsymbol{\varepsilon} \geq 0 \end{cases} . \tag{B.72}$$

Thus, we can obtain the dual form as below:

$$\begin{aligned} \min_{\boldsymbol{\mu}_1,\boldsymbol{\mu}_2} & \frac{1}{2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) - \boldsymbol{y}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\ \text{s.t.} \ & \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = 0 \\ & 0 \leq \boldsymbol{\mu}_1 \leq C \\ & 0 \leq \boldsymbol{\mu}_2 \leq C \\ & \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) \leq C_0\nu. \end{aligned} \tag{B.73}$$

The way of computing the optimal primal variables $\boldsymbol{w}$ and b is the same as in ε-SVR.

In addition to the properties of ε-SVR, ν-SVR has the following extra properties.

1. If $\nu > 1$, then $\varepsilon = 0$. This can be proven as follows. Because $\boldsymbol{\mu}_1 \leq \frac{C_0 \mathbf{1}}{n}$, we have $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}_1 \leq C_0$, that is $C_0 - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}_1 \geq 0$. If $\nu > 1$, then $C_0\nu - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}_1 > 0$. Because the KKT condition $C_0\nu - \mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) - \beta = 0$, therefore $\beta > 0$. From the complementary condition, we thus have $\varepsilon = 0$.

2. If $\nu > 0$, ν is an upper bound of the fraction of errors. Suppose the number of error is n_e, the number of upper-bounded multipliers is n_C. Because of the condition $\mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) \leq C_0\nu$, $n_C \leq \nu n$. Because $n_e \leq n_C \leq \nu n$, therefore $\frac{n_e}{n} \leq \nu$.

3. If $\nu > 0$, ν is a lower bound of the fraction of support vectors. Suppose the number of support vectors is n_S. Because $\nu > 0$, therefore $\beta = 0$. Then we have $\mathbf{1}^{\mathrm{T}}(\boldsymbol{\mu}_1 + \boldsymbol{\mu}_2) = C_0\nu$. If the multipliers of all support vectors are upper-bounded, then $n_S\frac{C_0}{n} = C_0\nu$, that is $n_S = \nu n$. Here, we use the property that either μ_{1i} or μ_{2i} must be zero. The multipliers of all support vectors are not necessarily upper-bounded, thus therefore $n_S \geq \nu n$, that is $\frac{n_S}{n} \geq \nu$.

2.8 l_2-Regularized Squared-Loss Linear Model

The squared loss can be applied in both classification and regression. The advantage of this loss function over those investigated above is its differentiability. However, we shall see that its disadvantage is that its formulation cannot be kernelized.

The primal form of considering the bias is

$$\min_{\boldsymbol{w},b} q(\boldsymbol{w}, b) = \frac{1}{2C}\|\boldsymbol{w}\|_2^2 + \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - b\|_2^2. \tag{B.74}$$

The above function can be rewritten as

$$q(\boldsymbol{w}, b) = \frac{1}{2C}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + \frac{1}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + b\mathbf{1}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} - \boldsymbol{y}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w} + \frac{n}{2}b^2 - b\mathbf{1}^{\mathrm{T}}\boldsymbol{y} + constant. \tag{B.75}$$

From the first-order optimality condition, we can have

$$\begin{cases} (\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} + \frac{1}{C}\boldsymbol{I})\boldsymbol{w} = \boldsymbol{X}(\boldsymbol{y} - b) \\ b = \frac{\mathbf{1}^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{w})}{n} \end{cases}. \tag{B.76}$$

In the above Equation, by substituting the second condition in the the righthand side the first condition, we have

$$\begin{aligned}\boldsymbol{X}(\boldsymbol{y}-b) &= \boldsymbol{X}(\boldsymbol{y}-\frac{\mathbf{1}^{\mathrm{T}}(\boldsymbol{y}-\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w})}{n}) \\ &= \boldsymbol{X}(\boldsymbol{y}-\frac{\mathbf{1}^{\mathrm{T}}}{n}\boldsymbol{y}+\frac{\mathbf{1}^{\mathrm{T}}}{n}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}) \\ &= \frac{1}{n}\boldsymbol{X}(n\boldsymbol{y}-\boldsymbol{E}\boldsymbol{y})+\frac{1}{n}\boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}, \end{aligned} \tag{B.77}$$

where $\boldsymbol{E}=\{1\}^{n\times n}$. Therefore, we have $(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}+\frac{1}{C}\boldsymbol{I})\boldsymbol{w}=\frac{1}{n}\boldsymbol{X}(n\boldsymbol{y}-\boldsymbol{E}\boldsymbol{y})+\frac{1}{n}\boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}$, that is

$$\boldsymbol{w}=(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}+\frac{1}{C}\boldsymbol{I}-\frac{1}{n}\boldsymbol{X}\boldsymbol{E}\boldsymbol{X}^{\mathrm{T}})^{-1}\boldsymbol{X}(\boldsymbol{y}-\frac{1}{n}\boldsymbol{E}\boldsymbol{y}). \tag{B.78}$$

After obtaining $\boldsymbol{w}$, we can compute the bias b using the second condition in Equation (B.76).

We have the following remarks regarding this model.

1. If we ignore the bias b, the problem expressed in Equation (B.74) is reduced to ridge regression. Then the solution becomes $\boldsymbol{w}=(\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}}+\frac{1}{C}\boldsymbol{I})^{-1}\boldsymbol{X}\boldsymbol{y}$.

2. From Equation (B.78), it can be seen that $\boldsymbol{w}$ can be solved by a closed-form equation.

3. However, $\boldsymbol{w}$ can not be represented by a function of inner products of the samples. Thus, this model can not be kernelized. Nevertheless, an explicit mapping function $\phi(\boldsymbol{x})$ can be used to project the samples from the input space to a high-dimensional feature space, where Equation (B.78) is solved explicitly.

4. The problem in Equation (B.74) is equivalent to

$$\begin{aligned}&\min_{\boldsymbol{w},b,\boldsymbol{\xi}} = \frac{1}{2C}\|\boldsymbol{w}\|_2^2+\frac{1}{2}\|\boldsymbol{\xi}\|_2^2 \\ &\text{s.t. } \boldsymbol{y}-\boldsymbol{X}^{\mathrm{T}}\boldsymbol{w}-b\mathbf{1}-\boldsymbol{\xi}=\mathbf{0}.\end{aligned} \tag{B.79}$$

   From the KKT conditions of this formulation, we can easily obtain that $\boldsymbol{w}=C\boldsymbol{X}\boldsymbol{\mu}$ (where $\boldsymbol{\mu}$ is the vector of Lagrange multipliers), that is $\boldsymbol{w}$ is a linear combination of the training samples. From Equation (B.78), we can see that the computation of $\boldsymbol{w}$ involves all training samples. Thus, $\boldsymbol{w}$ is not a *sparse* linear combination of the training samples. Thus, this model is not a sparse model. Unlike the hinge loss, l_1-loss, and ε-insensitive loss, the squared loss (l_2-loss) does not induce sparse error $\boldsymbol{\xi}$.

2.9 l_2-Regularized Logistic-Loss Linear Model

In addition to the squared loss, the logistic-loss is also differentiable. It can be applied to classification. The optimization of the model can be formulated as

$$\min_{\boldsymbol{w},b} f(\boldsymbol{w},b) = \frac{1}{2C}\|\boldsymbol{w}\|_2^2 + C\sum_{i=1}^{n}\log(1+e^{-\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}-y_i b}). \tag{B.80}$$

The partial derivative with respect to $\boldsymbol{w}$ is

$$\begin{aligned}\frac{\partial f}{\partial \boldsymbol{w}} &= \boldsymbol{w}^{\mathrm{T}} + C\sum_{i=1}^{n}[\frac{e^{-\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}-y_i b}}{1+e^{-\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}-y_i b}}(-\boldsymbol{z}_i^{\mathrm{T}})] \\ &= \boldsymbol{w}^{\mathrm{T}} - C\boldsymbol{p}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}},\end{aligned} \tag{B.81}$$

where $\boldsymbol{p}$ is defined as $p_i = \frac{1}{1+e^{\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}+y_i b}}$. The partial derivative with respect to b is defined as

$$\begin{aligned}\frac{\partial f}{\partial b} &= C\sum_{i=1}^{n}[\frac{e^{-\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}-y_i b}}{1+e^{-\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}-y_i b}}(-y_i)] \\ &= -C\boldsymbol{p}^{\mathrm{T}}\boldsymbol{y}.\end{aligned} \tag{B.82}$$

Therefore, the partial derivative with respect to $[\boldsymbol{w},b]^{\mathrm{T}}$ is

$$\frac{\partial f}{\partial [\boldsymbol{w};b]} = \left[\frac{\partial f}{\partial \boldsymbol{w}}, \frac{\partial f}{\partial b}\right] = \left[\boldsymbol{w}^{\mathrm{T}} - C\boldsymbol{p}^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}}, -C\boldsymbol{p}^{\mathrm{T}}\boldsymbol{y}\right] \tag{B.83}$$

The second derivative with respect to $[\boldsymbol{w},b]^{\mathrm{T}}$ can be written as

$$\frac{\partial^2 f}{\partial [\boldsymbol{w};b]^2} = \begin{bmatrix}\frac{\partial^2 f}{\partial \boldsymbol{w}^2}, \frac{\partial^2 f}{\partial b \partial \boldsymbol{w}} \\ \frac{\partial^2 f}{\partial \boldsymbol{w} \partial b}, \frac{\partial^2 f}{\partial b^2}\end{bmatrix} = \begin{bmatrix}\boldsymbol{I} + \boldsymbol{Z}\boldsymbol{Q}\boldsymbol{Z}^{\mathrm{T}} & (\boldsymbol{y}*\boldsymbol{q})^{\mathrm{T}}\boldsymbol{Z}^{\mathrm{T}} \\ \boldsymbol{Z}(\boldsymbol{y}*\boldsymbol{q}) & C\boldsymbol{m}^{\mathrm{T}}\boldsymbol{q}\end{bmatrix}, \tag{B.84}$$

where $\boldsymbol{Q} = \mathrm{diag}(\boldsymbol{q})$, and $\boldsymbol{q}$ is defined as $q_i = \frac{e^{\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}+y_i b}}{(1+e^{\boldsymbol{z}_i^{\mathrm{T}}\boldsymbol{w}+y_i b})^2}$, and $\boldsymbol{m} = \boldsymbol{y}*\boldsymbol{y}$. The chain rules of derivative, $\frac{\partial \boldsymbol{p}}{\partial \boldsymbol{w}} = -\boldsymbol{Q}\boldsymbol{Z}^{\mathrm{T}}$, and $\frac{\partial \boldsymbol{p}}{\partial b} = -\boldsymbol{q}*\boldsymbol{y}$ are used to derive the above equation. Once knowing the gradient and Hessian, we can resort to many numerical optimization methods to obtain the optimal solution of $\boldsymbol{w}$ and b.

2.10 Hypersphere One-Class SVM

The one-class classification problem is to identify outliers or novelties given limited number of training points. One-class SVM is an implementation of Vapnik's principle stating that we need to avoid solving more general problem than what is actually needed [243]. Instead of estimating the distribution of the data, one-class SVM simply estimates the boundary of the distribution which captures the main mass of the data. For this reason, one-class SVM is also called *support vector domain description* (SVDD) [154]. The border of the domain is defined by a non-negative linear combination of the outliers. SVDD finds the support of a multivariate distribution, where the support means the set of support vectors lying on and outside a desired boundary. Models differ in the shapes of the border. Tax and Duin treated it as a hypersphere [154]. And Schölkopf *et al.* treated it as a hyperplane [252]. Though looking quite different in primal form, they are equivalent under a weak condition, which can be seen in dual form. Both methods are introduced in this and the next sections.

The main idea of the hypersphere based SVDD, proposed by Tax and Duin [154], is in the following. Data points are implicitly mapped to a higher-dimensional feature space, where a hypersphere is learned such that its volume is as small as possible while keeps the core mass of the data. A hypersphere is defined by its center and (squared) radius. The hyperspheres in 1, 2, and 3-dimensional spaces are closed line, circle, and ball, respectively. An indicator function is learned such that the data points inside are positive (core data points), and these outside are negative (*outliers*).

This idea can be formulated, in primal form, as follows.

$$\begin{aligned} &\min_{R,\boldsymbol{\xi},\nu} \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \nu R \\ &\text{s.t. } \|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 \leq R + \xi_i, \quad i = 1, \cdots, n \\ &\qquad \boldsymbol{\xi} \geq 0 \\ &\qquad R > 0. \end{aligned} \tag{B.85}$$

where $\boldsymbol{c}$ is the center of the hypersphere, R is its squared radius, ξ_i is a slack variable

representing error, and vector $\boldsymbol{C}$ is constant with $C_i = \frac{1}{n}$. The Lagrange function is

$$\begin{aligned}
&L(R, \boldsymbol{\xi}, \boldsymbol{c}, \boldsymbol{\mu}, \boldsymbol{\eta}, \delta) \\
&= \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \nu R + \sum_{i=1}^{n} \mu_i(\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i) - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi} - \delta R \\
&= \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \nu R + \sum_{i=1}^{n} \mu_i\big(\phi^{\mathrm{T}}(\boldsymbol{x}_i)\phi(\boldsymbol{x}_i) - 2\phi^{\mathrm{T}}(\boldsymbol{x}_i)\boldsymbol{c} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{c} - R - \xi_i\big) - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi} - \delta R \\
&= \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} + \nu R + \boldsymbol{k}^{\mathrm{T}}\boldsymbol{\mu} - 2(\phi(\boldsymbol{X})\boldsymbol{\mu})^{\mathrm{T}}\boldsymbol{c} + s\boldsymbol{c}^{\mathrm{T}}\boldsymbol{c} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{R} - \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{\xi} - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi} - \delta R, \qquad \text{(B.86)}
\end{aligned}$$

where $\boldsymbol{k} = \mathrm{diag}(\phi(\boldsymbol{X})^{\mathrm{T}}\phi(\boldsymbol{X}))$, and $s = \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}$. Then, the KKT conditions are

$$\begin{cases}
\nu - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} - \delta = 0 \\
\boldsymbol{C} - \boldsymbol{\mu} - \boldsymbol{\eta} = \mathbf{0} \\
-\phi(\boldsymbol{X})\boldsymbol{\mu} + s\boldsymbol{c} = \mathbf{0} \\
\mu_i(\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i) = 0 \\
\boldsymbol{\eta} * \boldsymbol{\xi} = \mathbf{0} \\
\delta R = 0 \\
\boldsymbol{\mu} \geq 0 \\
\boldsymbol{\eta} \geq 0 \\
\delta \geq 0 \\
\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i \leq 0 \\
\xi_i \geq 0 \\
R > 0
\end{cases} . \qquad \text{(B.87)}$$

The dual form becomes

$$\begin{aligned}
&\min_{\boldsymbol{\mu}} \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu} - \frac{\nu}{2}\boldsymbol{k}^{\mathrm{T}}\boldsymbol{\mu} \qquad \text{(B.88)} \\
&\text{s.t. } \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = \nu \\
&\quad 0 \leq \boldsymbol{\mu} \leq C.
\end{aligned}$$

where $\boldsymbol{K} = \phi(\boldsymbol{X})^{\mathrm{T}}\phi(\boldsymbol{X})$, and $\boldsymbol{k} = \mathrm{diag}(\boldsymbol{K})$.

We denote $S = \{i|\mu_i > 0, i = 1, \cdots, n\}$ as the set of indices of nonzero multipliers which correspond to points on or outside the border. From the KKT conditions, we know that the

centroid of the hypersphere is a sparse non-negative linear combination of the training data points, that is $\boldsymbol{c} = \frac{1}{s}\phi(\boldsymbol{X})\boldsymbol{\mu} = \frac{1}{\nu}\phi(\boldsymbol{X})_S\boldsymbol{\mu}_S$. We define $B = \{i|0 < \mu_i < C, i = 1, \cdots, n\}$ as the subset of indices of points on the hypersphere. Then we can obtain R by the following formula:

$$R = \frac{1}{|B|}\sum_{b\in B}\|\phi(\boldsymbol{x})_b - \boldsymbol{c}\|_2^2 = \frac{1}{|B|}\Big(\text{trace}(\boldsymbol{K}_B) - \frac{2}{\nu}\text{sum}(\phi(\boldsymbol{X})_B^{\mathrm{T}}\phi(\boldsymbol{X})_S\boldsymbol{\mu}_S)\Big) + \frac{1}{\nu^2}\boldsymbol{\mu}_S^{\mathrm{T}}\boldsymbol{K}_S\boldsymbol{\mu}_S \tag{B.89}$$

The decision function is the following indicator function:

$$d(\boldsymbol{x}) = \text{sign}[f(\boldsymbol{x})], \tag{B.90}$$

where $f(\boldsymbol{x})$ is defined as below:

$$\begin{aligned} f(\boldsymbol{x}) &= R - \|\phi(\boldsymbol{x}) - \boldsymbol{c}\|_2^2 \\ &= R - \big(\phi^{\mathrm{T}}(\boldsymbol{x})\phi(\boldsymbol{x}) - 2\phi^{\mathrm{T}}(\boldsymbol{x})\boldsymbol{c} + \boldsymbol{c}^{\mathrm{T}}\boldsymbol{c}\big) \\ &= R - \big(\phi^{\mathrm{T}}(\boldsymbol{x})\phi(\boldsymbol{x}) - \frac{2}{\nu}\phi^{\mathrm{T}}(\boldsymbol{x})\phi(\boldsymbol{X})\boldsymbol{\mu} + \frac{1}{\nu^2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu}\big) \\ &= R - \big(\phi^{\mathrm{T}}(\boldsymbol{x})\phi(\boldsymbol{x}) - \frac{2}{\nu}\phi^{\mathrm{T}}(\boldsymbol{x})\phi(\boldsymbol{X})_S\boldsymbol{\mu}_S + \frac{1}{\nu^2}\boldsymbol{\mu}_S^{\mathrm{T}}\boldsymbol{K}_S\boldsymbol{\mu}_S\big). \end{aligned} \tag{B.91}$$

From the KKT conditions, we can obtain the following important properties.

1. If $\mu_i > 0$, the data point $\boldsymbol{x}_i$ resides either on or outside of the hypersphere. Such $\boldsymbol{x}_i$ corresponding to $\mu_i > 0$ is called a *support vector*. (We only call the points corresponding nonzero multipliers support vectors. From next point, we know that if $\boldsymbol{x}_i$ is on the hypersphere, then it is possible that $\mu_i = 0$.) Let's prove it in the following. Suppose $\mu_i > 0$. From complementarity condition $\mu_i(\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i) = 0$, we have $\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i = 0$. From KKT condition $C - \mu_i - \eta_i = 0$, we know $\eta_i \geq 0$. If $0 < \mu_i < C$, then $\eta_i > 0$, then from complementarity condition $\eta_i\xi_i = 0$, we have $\xi_i = 0$. Therefore $\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 = R$, that is $\boldsymbol{x}_i$ is on the hypersphere. If $\mu_i = C$, then $\xi_i \geq 0$, thus $\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 \geq R$. Thus, $\boldsymbol{x}_i$ is on or outside the hypersphere if $\mu_i = C$.

2. If $0 < \mu_i < C$, then data point $\boldsymbol{x}_i$ is on the hypersphere. However, the reverse is not true. We can only state that, if data point $\boldsymbol{x}_i$ is on the hypersphere, then $0 \leq \mu_i \leq C$. We can see this in the following. If $\boldsymbol{x}_i$ is on the hypersphere, then $\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R = 0$. From $\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i = 0$, we have $\xi_i = 0$. From $\eta_i\xi_i = 0$, we have $\eta_i \geq 0$. From $\eta_i = C - \mu_i$, we have $0 \leq \mu_i \leq C$. From $\mu_i(\|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_2^2 - R - \xi_i) = 0$, we know that

it is possible that $\mu_i = 0$.

3. If $\boldsymbol{x}_i$ resides outside of the hypersphere, then $\mu_i = C$. $\boldsymbol{x}_i$ is called an *outlier*. However, from the above, we can see that the reverse is not true.

4. As in the two-class ν-SVM, ν is a lower bound of the fraction of support vectors, and an upper bound of the fraction of outliers. That is $\frac{n_e}{n} \leq \nu \leq \frac{n_S}{n}$.

2.11 Hyperplane One-Class SVM

Schölkopf *et al.* propose to find a hyperplane rather than a hypersphere in the higher-dimensional feature space [252]. The hyperplane is defined as $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) - b$ $(b \geq 0)$. The indicator function $g(\boldsymbol{x}) = \mathrm{sign}[f(\boldsymbol{x})]$ takes $+1$ for a small region that captures most of the data, and -1 elsewhere. Because the distance from the origin to the hyperplane is $\frac{-b}{\boldsymbol{w}}$, therefore minimizing the negative distance is equivalent to maximizing the absolute distance (That is maximizing the margin). The objective task is therefore to minimize $\frac{1}{2}\|\boldsymbol{w}\|_2^2 - b$ as well as the loss.

It can be formulated as below:

$$\begin{aligned} \min_{\boldsymbol{w},b,\boldsymbol{\xi}} & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} - \nu b \\ \text{s.t.} & \phi(\boldsymbol{X})^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} + \boldsymbol{\xi} \geq 0 \\ & \boldsymbol{\xi} \geq 0 \\ & b > 0, \end{aligned} \tag{B.92}$$

where $\boldsymbol{C}$ is a constant vector with elements equal to $\frac{1}{n}$. The Lagrange function is

$$L(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\eta}, \delta) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} - \nu b - \boldsymbol{\mu}^{\mathrm{T}}(\phi(\boldsymbol{X})^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} + \boldsymbol{\xi}) - \boldsymbol{\eta}^{\mathrm{T}}\boldsymbol{\xi} - \delta b. \tag{B.93}$$

The KKT conditions are

$$\begin{cases} \boldsymbol{w} = \phi(\boldsymbol{X})\boldsymbol{\mu} \\ \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = \nu + \delta \\ \boldsymbol{C} - \boldsymbol{\mu} - \boldsymbol{\eta} = \mathbf{0} \\ \boldsymbol{\mu} * (\phi(\boldsymbol{X})^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} + \boldsymbol{\xi}) = \mathbf{0} \\ \boldsymbol{\eta} * \boldsymbol{\xi} = \mathbf{0} \\ \delta b = 0 \\ \boldsymbol{\mu} \geq 0 \\ \boldsymbol{\eta} \geq 0 \\ \delta \geq 0 \\ \phi(\boldsymbol{X})^{\mathrm{T}}\boldsymbol{w} - b\mathbf{1} + \boldsymbol{\xi} \geq 0 \\ \boldsymbol{\xi} \geq 0 \\ b > 0. \end{cases} \tag{B.94}$$

We can express the dual form as

$$\begin{aligned} \min_{\boldsymbol{\mu}} & \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu} \\ \text{s.t. } & \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = \nu \\ & 0 \leq \boldsymbol{\mu} \leq C. \end{aligned} \tag{B.95}$$

The decision function is thus

$$g(\phi(\boldsymbol{x})) = \operatorname{sign}[\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) - b] = \operatorname{sign}[\boldsymbol{\mu}_S^{\mathrm{T}}\phi(\boldsymbol{X}_S)^{\mathrm{T}}\phi(\boldsymbol{x}) - b], \tag{B.96}$$

where S is the set of indices of nonzero multipliers. In order to compute b, we need to find data points on the boundary. If $0 < \mu_i < C$, then data point $\boldsymbol{x}_i$ is on the bound and has $f(\boldsymbol{x}_i) = 0$. Therefore, we can find a set B that includes indices satisfying $0 < \mu_i < C$. Then, we compute b as

$$b = \operatorname{mean}(\boldsymbol{X}_B^{\mathrm{T}}\boldsymbol{X}_S\boldsymbol{\mu}_S). \tag{B.97}$$

The hyperplane based one-class SVM has the following important characteristics:

1. The relations between multiplier and position in hypersphere-based SVDD also apply

to this hyperplane-based SVDD.

2. As in the two-class SVM, it can be proven that ν is an upper bound of the fraction of outliers, and a lower bound of the fraction of support vectors, that is $\frac{n_e}{n} \leq \nu \leq \frac{n_S}{n}$.

3. ν equals $\frac{n_e}{n}$ and $\frac{n_S}{n}$ asymptotically with probability 1.

4. From the dual forms of both hypersphere and hyperplane formulations, we can see that the hypersphere formulation is equivalent to the hyperplane formulation in the case of constant $K(\boldsymbol{x}, \boldsymbol{x})$ because the linear term in the objective becomes constant.

3 A Review on Linear-Model Based Feature Selection

The sparse linear models can also be applied to select discriminative features. In this section, we review two main techniques. Both are based on the values of the weight vector $\boldsymbol{w}$.

3.1 l_1-norm SVM: l_1-Regularized Hinge-Loss Linear Model

l_1-norm SVM [129] is the maximum-margin classifier that uses l_1-norm regularization and hinge loss. It can be applied to simultaneous classification and variable selection. By virtue of the l_1-norm, the weight vector $\boldsymbol{w}$ becomes sparse. The basic idea is to select the features corresponding to non-negative weights. Essentially, this idea stems from LASSO [16].

The formulation of l_1-norm SVM is

$$\begin{aligned} &\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \|\boldsymbol{w}\|_1 + C\|\boldsymbol{\xi}\|_1 \\ &\text{s.t. } \boldsymbol{Z}^{\mathrm{T}}\boldsymbol{w} + b\boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi} \\ &\qquad \boldsymbol{\xi} \geq 0. \end{aligned} \tag{B.98}$$

The first term in the objective is not differentiable. The second term is differentiable due to its non-negative constraint. $\boldsymbol{w}$ can be decomposed by $\boldsymbol{w} = \boldsymbol{w}_+ - \boldsymbol{w}_-$ where $\boldsymbol{w}_+$ is defined as $w_{+i} = w_i$ if $w_i \geq 0, w_{+i} = 0$ otherwise. And $\boldsymbol{w}_-$ is defined as $w_{-i} = -w_i$ if $w_i \leq 0, w_{-i} = 0$ otherwise. Similarly b can be decomposed by $b = b_+ - b_-$, where $b_+, b_- \geq 0$.

The optimization is then equivalent to

$$\begin{aligned} \min_{\boldsymbol{w}_+,\boldsymbol{w}_-,b_+,b_-,\boldsymbol{\xi}} & \mathbf{1}^{\mathrm{T}}\boldsymbol{w}_+ + \mathbf{1}^{\mathrm{T}}\boldsymbol{w}_- + \boldsymbol{C}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t. } & \boldsymbol{Z}^{\mathrm{T}}(\boldsymbol{w}_+ - \boldsymbol{w}_-) + (b_+ - b_-)\boldsymbol{y} \geq \mathbf{1} - \boldsymbol{\xi} \\ & \boldsymbol{w}_+ \geq 0 \\ & \boldsymbol{w}_- \geq 0 \\ & b_+, b_- \geq 0 \\ & \boldsymbol{\xi} \geq 0. \end{aligned} \tag{B.99}$$

We can rewrite it into standard form:

$$\begin{aligned} \min_{\boldsymbol{w}_+,\boldsymbol{w}_-,b_+,b_-,\boldsymbol{\xi}} & [\mathbf{1};\mathbf{1};\delta,\delta,\boldsymbol{C}]^{\mathrm{T}} \begin{bmatrix} \boldsymbol{w}_+ \\ \boldsymbol{w}_- \\ b_+ \\ b_- \\ \boldsymbol{\xi} \end{bmatrix} \\ \text{s.t. } & [-\boldsymbol{Z}^{\mathrm{T}}, \boldsymbol{Z}^{\mathrm{T}}, -\boldsymbol{y}, \boldsymbol{y}, -\boldsymbol{I}] \begin{bmatrix} \boldsymbol{w}_+ \\ \boldsymbol{w}_- \\ b_+ \\ b_- \\ \boldsymbol{\xi} \end{bmatrix} \leq -\mathbf{1} \\ & \boldsymbol{w}_+ \geq 0, \boldsymbol{w}_- \geq 0 \\ & b_+, b_- \geq 0 \\ & \boldsymbol{\xi} \geq 0, \end{aligned} \tag{B.100}$$

where δ is a very small positive constant, e.g. 10^{-10}, to avoid numeric error. We can find that the optimization is a large-scale *linear programming* (LP) problem.

After obtaining the sparse $\boldsymbol{w}$, we can take only the features corresponding to non-zeros. Let's define $\mathcal{V} = \{i|w_i \neq 0\}$ being indices of non-zeros weights, the linear function can be written as

$$f(\boldsymbol{x}) = \boldsymbol{x}_{\mathcal{V}}^{\mathrm{T}}\boldsymbol{w}_{\mathcal{V}} + b. \tag{B.101}$$

3.2 SVM-Recursive Feature Elimination [1]

In bioinformatics, *support vector machine recursive feature elimination* (SVM-RFE) is a successful method for gene selection [127, 51]. SVM-RFE only uses support vectors to rank features, which is an idea of combining sample selection in feature selection because SVM-RFE selects the boundary samples simultaneously. (There are also some other ideas that prototypic samples are selected. Interested reader are referred to [51] for a concise review of sample selection and [130] from another viewpoint.)

SVM-RFE can be viewed as both feature ranking method and feature subset selection method. Taking Algorithm B.1 in gene selection for example, if the first step (backward search) is only used to sort genes, it is a ranking method; whereas if it involves forward search after backward search to include the sorted genes one by one until the classification performance degenerates, then it is a gene subset selection method. SVM-RFE does not fix the size of gene subset. Mundra and Rajapakse combined the *minimum Redundancy Maximum Relevance Feature Selection* (mRMR) measure with SVM-RFE (SVM-RFE-mRMR) [253], and reported better accuracy than the original mRMR and SVM-RFE methods.

SVM-RFE may have two issues in practice. First, if the current best validation accuracy in the validation step meets 1, SVM-RFE may continue adding features in the subset until the current validation accuracy is less than 1. For instance, the sequence of the best validation accuracy is $[0.6, 0.8, 1, 1, 1, 0.9]$ and the sorted features in ascent order is $[\cdots, g_8, g_3, g_{10}, g_2, g_9, g_6]$, SVM-RFE may return $[g_6, g_9, g_2, g_{10}, g_3]$, but the algorithm should terminate at the third iterations and return $[g_6, g_9, g_2]$. Second, if the current best validation accuracy is less than 1, and this is unchanged until the current validation accuracy is less than it. SVM-RFE may keep adding features before this. Let us use the above example. If we change 1 to 0.95, similarly SVM-RFE may return $[g_6, g_9, g_2, g_{10}, g_3]$. Moreover, since SVM-RFE uses a variant of backward search, if the number of features is very large, it is too computationally expensive to apply in practice. We revised the SVM-RFE method to solve these weaknesses. [127] and [253] only described the feature ranking step, which is actually incomplete, we therefore append the validation step to find the optimal feature subset. This turns out the complete SVM-RFE method as described in Algorithm B.1.

4 A Review on The Decomposition Methods for SVMs

As can be seen above that, the dual forms of many regularized linear models are constrained QP problems. For a large number of training samples, solving the large-scale QP problems

[1] The content of this section is based on our publication [128].

Algorithm B.1 *Revised SVM-RFE Feature Subset Selection*

Input: $\boldsymbol{X}$, of size m(features) $\times$ n(samples), and the class labels $\boldsymbol{y}$

Output: selected feature subset $\boldsymbol{g}$, the best validation accuracy av,and list of survived features $\boldsymbol{f}$

Split $\boldsymbol{X}$ into training set $\boldsymbol{X}_{\text{trace}}$ and validation set $\boldsymbol{X}_{\text{val}}$;
May filter out the features over $\boldsymbol{X}_{\text{trace}}$, and get feature list $\boldsymbol{f}$ left;
$\boldsymbol{X}_{\text{trace}} = \boldsymbol{X}_{\text{trace}}(\boldsymbol{f}, :)$;
$\boldsymbol{X}_{\text{val}} = \boldsymbol{X}_{\text{val}}(\boldsymbol{f}, :)$;
————————gene ranking step————————
Given set of features $\boldsymbol{s}$ initially including all features;
Ranked set of features, $\boldsymbol{r} = \{\}$;
repeat
  Train a linear SVM over $\boldsymbol{X}_{\text{trace}}$ with feature set $\boldsymbol{s}$;
  Calculate the weight of each feature w_i;
  for each gene $i \in \boldsymbol{s}$ **do**
    Compute $r_i = |w_i|$;
  end for
  Select the feature with smallest ranking score, $i^* = \arg\min \{r_i\}$;
  Update $\boldsymbol{r} = \boldsymbol{r} \cup \{i^*\}$; $\boldsymbol{s} = \boldsymbol{s} \setminus \{i^*\}$;
until all features are ranked
————————validation step————————
$\boldsymbol{g} = \{\}$;
Set the best validation accuracy $av = 0$;
for i=length($\boldsymbol{r}$) **to** 1 **do**
  $\boldsymbol{s} = \boldsymbol{s} \cup \{r_i\}$;
  Train a linear SVM classifier over $\boldsymbol{X}_{\text{trace}}$;
  Obtain the validation accuracy a through validating the classifier over $\boldsymbol{X}_{\text{val}}$;
  if $av \leq a$ **then**
    if $av < a$ **then**
      $\boldsymbol{g} = \boldsymbol{s}$
    end if
    if $av == 1$ **then**
      Break;
    end if
  else
    Break;
  end if
end for

is challenging. One idea of solving this issue is dividing one QP problem into smaller ones. Well-known methods based on this idea include chunking method [254], decomposition

method [255, 28] and *sequential minimal optimization* (SMO) method [256, 30]. The idea of decomposition method is that only a few variables are selected to update, while keeping the rest unchanged. The decomposition method usually makes use of the fact that the dual form of a SVM is constrained by one equality and the inequalities are only boundary constraints. SMO is a specific case of the decomposition method. In each iteration, it only selects the minimal number of variables as free variables and fixes all other variables. In C-SVM and ν-SVM, the minimal numbers of variables are two and three, respectively. Its advantages are in the following. First, analytical solution can be obtained rather than invoking a numerical QP solver. Second, it may not require matrix storage. In the following, we shall describe the formulations of decomposition method for C-SVM, ν-SVM, and hypersphere-SVDD. Their corresponding SMO methods are given in details.

4.1 Decomposition Method for C-SVM

First of all, we formulate the decomposition method for solving the dual form of the C-SVM (Equation (B.39)). We define the kernel matrix as $\boldsymbol{K} = [k_{ij} = k(\boldsymbol{x}_i, \phi(\boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)]$, and $\bar{\boldsymbol{K}} = [\bar{k}_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)]$, where $k(\cdot,\cdot)$ is an explicit kernel function, and $\phi(\cdot)$ is the corresponding implicit basis mapping function. We also assume the kernel function is symmetric, that is $k(\boldsymbol{x}_i, \phi(\boldsymbol{x}_j) = k(\boldsymbol{x}_j, \phi(\boldsymbol{x}_i)$. Let A be a working set containing the variables to be updated, and P be a fixed set including the reminding variables. Then, the decomposition of the objective of the dual form becomes

$$\begin{aligned} g(\boldsymbol{\mu}_A) &= [\boldsymbol{\mu}_A^{\mathrm{T}}, \boldsymbol{\mu}_P^{\mathrm{T}}] \begin{bmatrix} \bar{\boldsymbol{K}}_{AA} & \bar{\boldsymbol{K}}_{AP} \\ \bar{\boldsymbol{K}}_{PA} & \bar{\boldsymbol{K}}_{PP} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_P \end{bmatrix} - [\mathbf{1}_A^{\mathrm{T}}, \mathbf{1}_P^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_P \end{bmatrix} \\ &= \frac{1}{2}(\boldsymbol{\mu}_A^{\mathrm{T}}\bar{\boldsymbol{K}}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PP}\boldsymbol{\mu}_P) - \mathbf{1}_A^{\mathrm{T}}\boldsymbol{\mu}_A - \mathbf{1}_P^{\mathrm{T}}\boldsymbol{\mu}_P \\ &= \frac{1}{2}\boldsymbol{\mu}_A^{\mathrm{T}}\bar{\boldsymbol{K}}_{AA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PA}\boldsymbol{\mu}_A - \mathbf{1}_A^{\mathrm{T}}\boldsymbol{\mu}_A + constant, \end{aligned} \tag{B.102}$$

where $\bar{\boldsymbol{K}}_{AP}$ is a submatrix of $\bar{\boldsymbol{K}}$ obtained via taking the rows corresponding to A and columns corresponding to P. The equality constraint of Equation (B.39) can be decomposed into

$$\boldsymbol{y}_A^{\mathrm{T}}\boldsymbol{\mu}_A = -\boldsymbol{y}_P^{\mathrm{T}}\boldsymbol{\mu}_P. \tag{B.103}$$

Thus, the reduced problem with respect to $\boldsymbol{\mu}_A$ is

$$\begin{aligned} \min_{\boldsymbol{\mu}_A} & g(\boldsymbol{\mu}_A) = \frac{1}{2}\boldsymbol{\mu}_A^{\mathrm{T}}\bar{\boldsymbol{K}}_{AA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PA}\boldsymbol{\mu}_A - \mathbf{1}_A^{\mathrm{T}}\boldsymbol{\mu}_A \\ \text{s.t. } & \boldsymbol{y}_A^{\mathrm{T}}\boldsymbol{\mu}_A = -\boldsymbol{y}_P^{\mathrm{T}}\boldsymbol{\mu}_P \\ & 0 \leq \boldsymbol{\mu}_A \leq C. \end{aligned} \tag{B.104}$$

The optimization of this decomposition method proceeds iteratively. In each iteration, some variables are firstly selected to form A; and then a QP solver is invoked to solve Equation (B.104) and update the variables $\boldsymbol{\mu}_A$. The implementations of the decomposition method differ from the way of selecting working variables [257].

4.2 SMO Method for C-SVM

Our following derivation of the SMO method for C-SVM is based on Platt's work [256, 30]. From the decomposed equality in Equation (B.103), one can find that the minimal number of working variables should be two. This is because if there is only one working variable, the value of this working variable can be obtained from the constraint without looking at the objective. With loss of generality, we use the convention of [256] that representing the two working variables by x_1 and x_2. We define $k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\phi(\boldsymbol{x}_i))^{\mathrm{T}}\phi(\boldsymbol{x}_j)$. The reduced problem with two working variables can be expressed as

$$\begin{aligned} \min_{\mu_1,\mu_2} & g(\mu_1, \mu_2) = \frac{1}{2}(k_{11}\mu_1^2 + 2y_1y_2k_{12}\mu_1\mu_2 + k_{22}\mu_2^2) + v_1\mu_1 + v_2\mu_2 \\ \text{s.t. } & \mu_1 + \mu_2 = t \\ & 0 \leq \mu_1, \mu_2 \leq C, \end{aligned} \tag{B.105}$$

where we denote $v_i = y_i(\boldsymbol{y}_P * \boldsymbol{\mu}_P)^{\mathrm{T}}\boldsymbol{K}_{Pi} - 1$ (where P contains the remaining $n-2$ variables fixed), and $t = -\boldsymbol{y}_P^{\mathrm{T}}\boldsymbol{\mu}_P$.

We do not want to compute v_i directly, because it takes linear time. Instead, we can compute it by using the previous value of the linear function $u_i' = f(\boldsymbol{x}_i)$, which takes constant time. The previous value u_i' can be decomposed as

$$u_i' = \boldsymbol{w}'^{\mathrm{T}}\boldsymbol{x}_i + b' = (\boldsymbol{\mu}' * \boldsymbol{y})^{\mathrm{T}}\boldsymbol{K}_{:,i} + b = \mu_1'y_1k_{1i} + \mu_2'y_2k_{2i} + (\boldsymbol{\mu}_P' * \boldsymbol{y}_P)^{\mathrm{T}}\boldsymbol{K}_{P,i} + b, \tag{B.106}$$

where $\boldsymbol{w}'$ is the previous value of $\boldsymbol{w}$, and similarly for the other variables, we use the prime

symbol to denote the previous value of a variable. Therefore we have

$$(\boldsymbol{\mu}'_P * \boldsymbol{y}_P)^{\mathrm{T}} \boldsymbol{K}_{P,i} = u'_i - \mu'_1 y_1 k_{1i} - \mu'_2 y_2 k_{2i} - b. \tag{B.107}$$

Therefore

$$v_i = y_i u'_i - y_1 y_i k_{1i} \mu'_1 - y_2 y_i k_{2i} \mu'_2 - y_i b - 1. \tag{B.108}$$

Thus, we can represent v_1 and v_2 in constant time as follows

$$v_1 = y_1 u'_1 - y_1 y_1 k_{11} \mu'_1 - y_1 y_2 k_{12} \mu'_2 - y_1 b - 1, \tag{B.109}$$

$$v_2 = y_2 u'_2 - y_1 y_2 k_{12} \mu'_1 - y_2 y_2 k_{22} \mu'_2 - y_2 b - 1. \tag{B.110}$$

Similarly, we do not want to compute t directly, as it takes linear time, we can instead compute it in constant time by using the previous values μ'_1 and μ'_2:

$$\mu_1 + y_1 y_2 \mu_2 = \mu'_1 + y_1 y_2 \mu'_2 = t. \tag{B.111}$$

If the second derivative of the objective in Equation (B.105) along the line defined by the equality is non-negative, then the problem has a minimum solution. In the general decomposition method, the derivative of $f(\boldsymbol{\mu}_A)$ with respect to $\boldsymbol{\mu}_A$ is

$$\frac{\partial f(\boldsymbol{\mu}_A)}{\boldsymbol{\mu}_A} = \boldsymbol{\mu}_A^{\mathrm{T}} \bar{\boldsymbol{K}}_{AA} + \boldsymbol{\mu}_P^{\mathrm{T}} \bar{\boldsymbol{K}}_{PA} - \mathbf{1}^{\mathrm{T}}. \tag{B.112}$$

Therefore, the first derivative of $f(\boldsymbol{\mu}_A)$ along the direction $\boldsymbol{d}$ is

$$\begin{aligned} \frac{\partial f(\boldsymbol{\mu}_A)}{\boldsymbol{d}} &= \frac{\partial f(\boldsymbol{\mu}_A)}{\boldsymbol{\mu}_A} \boldsymbol{d} \\ &= (\boldsymbol{\mu}_A^{\mathrm{T}} \bar{\boldsymbol{K}}_{AA} + \boldsymbol{\mu}_P^{\mathrm{T}} \bar{\boldsymbol{K}}_{PA} - \mathbf{1}^{\mathrm{T}}) \boldsymbol{d}. \end{aligned} \tag{B.113}$$

The second derivative of $f(\boldsymbol{\mu}_A)$ along the direction $\boldsymbol{d}$ is

$$\begin{aligned} \frac{\partial f^2(\boldsymbol{\mu}_A)}{\boldsymbol{d}^2} &= \frac{\partial \frac{\partial f(\boldsymbol{\mu}_A)}{\partial \boldsymbol{d}}}{\partial \boldsymbol{\mu}_A} \boldsymbol{d} \\ &= \boldsymbol{d}^{\mathrm{T}} \bar{\boldsymbol{K}}_{AA} \boldsymbol{d}. \end{aligned} \tag{B.114}$$

For the two-variables case, $\boldsymbol{d} = [y_2; -y_1]$, therefore the second derivative of $f([\mu_1, \mu_2])$ along the linear line defined by $y_1\mu_1 + y_2\mu_2 = constant$ is $k_{11} + k_{22} - 2k_{12}$.

Solve the Two-Variables Problem

In order to solve the problem (Equation (B.105)), we solve μ_2 first, then μ_1. We substitute $\mu_1 = t - y_1y_2\mu_2$ in the objective:

$$g(\mu_2) = \frac{1}{2}(k_{11}(t - y_1y_2\mu_2) + 2y_1y_2k_{12}(t - y_1y_2\mu_2)\mu_2 + k_{22}\mu_2^2) + v_1(t - y_1y_2\mu_2) + v_2\mu_2. \tag{B.115}$$

$\mu_1 = t - y_1y_2\mu_2$ needs also to be substituted in the inequality constraints in order to let μ_1 disappear in the constraints:

$$\begin{cases} 0 \leq t - y_1y_2\mu_2 \leq C \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.116}$$

If $y_1y_2 = 1$, we have

$$\begin{cases} 0 \leq t - \mu_2 \leq C \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.117}$$

That is

$$\begin{cases} t - C \leq \mu_2 \leq t \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.118}$$

That is

$$L = \max(0, t - C), \quad U = \min(C, t), \tag{B.119}$$

where $t = \mu_1' + \mu_2'$. If $y_1y_2 = -1$, we have

$$\begin{cases} 0 \leq t + \mu_2 \leq C \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.120}$$

That is

$$\begin{cases} -t \leq \mu_2 \leq C - t \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.121}$$

That is

$$L = \max(0, -t), \quad U = \min(C, C - t), \tag{B.122}$$

where $t = \mu_1' - \mu_2'$. Therefore, the optimization with respect to μ_2 is

$$\min_{\mu_2} g(\mu_2) = \frac{1}{2}(k_{11}(t - y_1 y_2 \mu_2) + 2y_1 y_2 k_{12}(t - y_1 y_2 \mu_2)\mu_2 + k_{22}\mu_2^2) + v_1(t - y_1 y_2 \mu_2) + v_2 \mu_2 \tag{B.123}$$

s.t. $L \leq \mu_2 \leq U$.

Taking derivative with respect to μ_2, and letting it to be zero, we have

$$\frac{\partial g(\mu_2)}{\mu_2} = (k_{11} + k_{22} - 2k_{12})\mu_2 - y_1 y_2 k_{11} t + y_1 y_2 k_{12} t - y_1 y_2 v_1 + v_2 = 0. \tag{B.124}$$

Thus, we have the optimal μ_2:

$$\mu_2 = \frac{y_1 y_2 k_{11} t - y_1 y_2 k_{12} t + y_1 y_2 v_1 - v_2}{(k_{11} + k_{22} - 2k_{12})}. \tag{B.125}$$

For further convenience, we substitute v_1, v_2, and t in it and obtain

$$\mu_2 = \mu_2' + \frac{y_2((u_1' - y_1) - (u_2' - y_2))}{(k_{11} + k_{22} - 2k_{12})}. \tag{B.126}$$

Considering the constraint with respect to μ_2, the clipped optimal solution is

$$\mu_2 = \begin{cases} U & \text{if } \mu_2 \geq U \\ \mu_2 & \text{if } L < \mu_2 < U \\ L & \text{if } \mu_2 \leq L. \end{cases} \tag{B.127}$$

Using the equality constraint, we can obtain the solution to μ_1:

$$\mu_1 = t - y_1 y_2 \mu_2 = \mu_1' + y_1 y_2(\mu_2' - \mu_2), \tag{B.128}$$

where μ_1 satisfies $0 \leq \mu_1 \leq C$.

Update the Bias

After updating μ_1 and μ_2, the bias b should be updated accordingly. We know that there are two common methods to compute the bias. The first method is to use $b = y_i - \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}_i) = y_i - \boldsymbol{K}_{i,:}(\boldsymbol{\mu} * \boldsymbol{y})$ where the corresponding $0 < \mu_i < C$. The second method is more robust, it compute the average bias of all support vectors corresponding to non-bounded multipliers. Both methods takes linear time complexity. Alternatively, we can update b in constant time by using the previous output. If $0 < \mu_1 < C$, then the output of $f(\boldsymbol{x})$ is expected to y_1. Therefore $y_1 = \boldsymbol{K}_{1,:}(\boldsymbol{\mu} * \boldsymbol{y}) + b_1$. We know that $u_1' = \boldsymbol{K}_{1,:}(\boldsymbol{\mu}' * \boldsymbol{y}) + b'$. Taking the difference of both equations, we have $y_1 - u_1' = y_1 k_1 1(\mu_1 - \mu_1') + y_2 k_1 2(\mu_2 - \mu_2') + b_1 - b'$. Therefore we have

$$b_1 = y_1 - u_1' + y_1 k_{11}(\mu_1' - \mu_1) + y_2 k_{12}(\mu_2' - \mu_2) + b'. \tag{B.129}$$

Similarly, if $0 < \mu_2 < C$, we have

$$b_2 = y_2 - u_2' + y_1 k_{21}(\mu_1' - \mu_1) + y_2 k_{22}(\mu_2' - \mu_2) + b'. \tag{B.130}$$

If both μ_1 and μ_2 are at bound, that is either 0 or C, we need to discuss them in different conditions. The rationale behind this is that the bias does not need to updated if they are identical with their previous values. If the values of μ_1 and μ_2 result in $L = U$, then we could also determine that $\mu_1 = \mu_1'$ and $\mu_2 = \mu_2'$, therefore there is no need to update the bias. Furthermore, this situation should be avoided earlier when selecting two working variables which violate the KKT condition. If both μ_1 and μ_2 are at bound and $L \neq U$, then any bias between b_1 and b_2 satisfies the KKT condition. This is because we need to shift the fixed margin to let both points not sit outside it. Platt's method is simply choose the half way. We summarize the updates of b under different situations in Table B.1.

Update Output

After updating μ_1 and μ_2, all the output $u_i = f(\boldsymbol{x}_i)$ should be updated because it is a function of $\boldsymbol{\mu}$. We can update each in constant time:

$$u_i = u_i' + y_1(\mu_1 - \mu_1')k_{1i} + y_2(\mu_2 - \mu_2')k_{2i} + b - b'. \tag{B.131}$$

Table B.1: Updates b in different situations.

μ_1	μ_2	y_1y_2	b
$0 < \mu_1 < C$	-	-	b_1
		-	b_1
-	$0 < \mu_2 < C$	-	b_2
		-	b_2
$0 < \mu_1 < C$	$0 < \mu_2 < C$	-	$\frac{b_1+b_2}{2}$
		-	$\frac{b_1+b_2}{2}$
0	0	$y_1y_2 = 1, t = 0, L = 0, U = 0$	b'
		$y_1y_2 = -1, t = 0, L = 0, U = C$	$\frac{b_1+b_2}{2}$
C	C	$y_1y_2 = 1, t = 2C, L = C, U = C$	b'
		$y_1y_2 = -1, t = 0, L = 0, U = C$	$\frac{b_1+b_2}{2}$
0	C	$y_1y_2 = 1, t = C, L = 0, U = C$	$\frac{b_1+b_2}{2}$
		$y_1y_2 = -1, t = -C, L = C, U = C$	b'
C	0	$y_1y_2 = 1, t = C, L = 0, U = C$	$\frac{b_1+b_2}{2}$
		$y_1y_2 = -1, t = C, L = 0, U = 0$	b'

Select Two Working Variables

We need heuristic to select two working variables. The basis method is to choose μ_1 which violates the KKT condition, and then choose μ_2 which obtains the largest $|(u_1' - y_1) - (u_2' - y_2)|$. Let's define N the set indices of non-bound multipliers, B the set of indices of upper bound multipliers, and S the set of indices of nonzero multipliers (hence $S = N \cup B$). The heuristic first select μ_1 from $\boldsymbol{\mu}_B$, and then select μ_2 from $\boldsymbol{\mu}$ to update both multipliers. After this update, the algorithm iterates on $\boldsymbol{\mu}_N$ to find μ_1 until all samples corresponding to N satisfy the KKT conditions. Then the algorithm repeats the above two step until all samples in $\boldsymbol{X}$ satisfy the KKT conditions.

From table B.1, we know that, if $(y_iy_j = 1 \cap (\mu_i' = \mu_j' = 0 \cup \mu_i' = \mu_j' = C) \cup (y_iy_j = -1 \cap ((\mu_i' = 0 \cap \mu_j' = C) \cup (\mu_i' = C \cap \mu_j' = 0)))$ (this condition is equivalent to $L = U$), μ_i and μ_j should never be selected, because the new values of this two variables would not change after optimization.

Initialization

The large QP problem needs to be initialized quickly by a feasible solution which fulfils the equality and bound constraints. One simple method is to let the values of s Lagrange multipliers, corresponding to positive and negative training samples, be a constant a where $0 < a < C$, respectively, and set the values of the remaining multipliers to 0.

4.3 Decomposition Method for ν-SVM

Now, we derive the decomposition method for the dual form of ν-SVM (Equation (B.46)). We define $\boldsymbol{K} = [k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)]$, and $\bar{\boldsymbol{K}} = [\bar{k}_{ij} = y_i y_j \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)]$. In the literature, the inequality constraint $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} \geq \nu$ is usually replaced with the equality constraint $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = \nu$ for the convenience of optimization [258, 164]. Consequently, the optimization becomes:

$$\begin{aligned} \min_{\boldsymbol{\mu}} & g(\boldsymbol{\mu}) = \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\bar{\boldsymbol{K}}\boldsymbol{\mu} \\ \text{s.t. } & \boldsymbol{y}^{\mathrm{T}}\boldsymbol{\mu} = 0 \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu} = \nu \\ & 0 \leq \boldsymbol{\mu} \leq \boldsymbol{C}, \end{aligned} \tag{B.132}$$

where $C = \frac{1}{n}$.

Let A be the working set, and P be fixed set. Then the decomposition of the objective of the dual form becomes

$$\begin{aligned} g(\boldsymbol{\mu}_A) &= [\boldsymbol{\mu}_A^{\mathrm{T}}, \boldsymbol{\mu}_P^{\mathrm{T}}] \begin{bmatrix} \bar{\boldsymbol{K}}_{AA} & \bar{\boldsymbol{K}}_{AP} \\ \bar{\boldsymbol{K}}_{PA} & \bar{\boldsymbol{K}}_{PP} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_P \end{bmatrix} \\ &= \frac{1}{2}(\boldsymbol{\mu}_A^{\mathrm{T}}\bar{\boldsymbol{K}}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PP}\boldsymbol{\mu}_P) \\ &= \frac{1}{2}\boldsymbol{\mu}_A^{\mathrm{T}}\bar{\boldsymbol{K}}_{AA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\bar{\boldsymbol{K}}_{PA}\boldsymbol{\mu}_A + \textit{constant}. \end{aligned} \tag{B.133}$$

The equality constraints can be decomposed into $\boldsymbol{y}_A^{\mathrm{T}}\boldsymbol{\mu}_A = -\boldsymbol{y}_P^{\mathrm{T}}\boldsymbol{\mu}_P$, and $\mathbf{1}_A^{\mathrm{T}}\boldsymbol{\mu}_A = \nu - \mathbf{1}_P^{\mathrm{T}}\boldsymbol{\mu}_P$.

4.4 SMO Method for ν-SVM

The authors of [164] state that the minimal number of working variables is two for solving ν-SVM by decomposition method. However, we argue that at least three variables must be required, because when using only two variables, we have problem to optimize the reduced optimization or update b. In the following, we prove that two working variables are impossible for SMO to solve ν-SVM.

Proof. The objective of the two-variables problem is

$$\begin{aligned} &\min_{\mu_1,\mu_2} g(\mu_1,\mu_2) = \frac{1}{2}(k_{11}\mu_1^2 + 2y_1y_2k_{12}\mu_1\mu_2 + k_{22}\mu_2^2) + y_1v_1\mu_1 + y_2v_2\mu_2 \\ &\text{s.t. } \mu_1 + y_1y_2\mu_2 = t_1 \\ &\qquad \mu_1 + \mu_2 = t_2 \\ &\qquad 0 \le \mu_1, \mu_2 \le C, \end{aligned} \tag{B.134}$$

where $v_i = (\boldsymbol{\mu}_P * \boldsymbol{y}_P)^{\mathrm{T}}\boldsymbol{K}_{Pi}$, $t_1 = \mu_1' + y_1y_2\mu_2'$, and $t_2 = \mu_1' + \mu_2'$. The prime notation indicates the previous value of a variable. We have two variables and two equality constraints. If $y_1y_2 = -1$, then there is only one unique solution to the equality constraints. The unique solution is in fact the previous values μ_1' and μ_2'. Therefore, the optimization would not change the values of the multipliers. We hence have to avoid choosing two variables having different class labels. If $y_1y_2 = 1$, then the two equality constraints coincide into one $\mu_1 + \mu_2 = t$, where $t = t_1 = t_2 = \mu_1' + \mu_2'$. There are infinite number feasible solutions in this situation. When $y_1 = y_2$, the optimization is simplified into

$$\begin{aligned} &\min_{\mu_1,\mu_2} g(\mu_1,\mu_2) = \frac{1}{2}(k_{11}\mu_1^2 + 2y_1y_2k_{12}\mu_1\mu_2 + k_{22}\mu_2^2) + y_1v_1\mu_1 + y_2v_2\mu_2 \\ &\text{s.t. } \mu_1 + \mu_2 = t \\ &\qquad 0 \le \mu_1, \mu_2 \le C. \end{aligned} \tag{B.135}$$

v_i can be computed by $u_i' = f(\boldsymbol{x}_i|\boldsymbol{w}', b') = \boldsymbol{k}_{i,:}(\boldsymbol{y} * \boldsymbol{\mu}') + b'$:

$$v_i = u_i - y_1\mu_1'k_{1i} - y_2\mu_2'k_{2i} - b'. \tag{B.136}$$

As has been discussed in Section 2.4, we know that updating b needs two training samples from different classes, that is $y_1y_2 = -1$. From the above derivation, we also know that $y_1y_2 = -1$ is impossible for optimization. Therefore, we can state that more than two working variables are needed for SMO. □

If we select three working variables such that $y_1 = y_2$, and $y_3 \neq y_1$, then we can avoid the problems of optimization and updating b. Therefore, three variables are required for SMO to solve ν-SVM. The idea of solving ν-SVM by SMO is similar with, but more complex than, that of solving C-SVM. Thus, we decide to omit the detail.

4.5 Decomposition Method for Hypersphere One-Class SVM

The dual form of hypersphere SVDD has been given in Equation (B.88). Now, we define $\boldsymbol{K} = [k_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}_j)]$, and $\boldsymbol{k} = \mathrm{diag}(\boldsymbol{K})$. Let $\boldsymbol{q} = \frac{\nu}{2}\boldsymbol{k}$. Let A be the working set, and P be fixed set. Then the decomposition of the objective of the dual form becomes

$$\begin{aligned} g(\boldsymbol{\mu}_A) &= [\boldsymbol{\mu}_A^{\mathrm{T}}, \boldsymbol{\mu}_P^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{K}_{AA} & \boldsymbol{K}_{AP} \\ \boldsymbol{K}_{PA} & \boldsymbol{K}_{PP} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_P \end{bmatrix} - [\boldsymbol{q}_A^{\mathrm{T}}, \boldsymbol{q}_P^{\mathrm{T}}] \begin{bmatrix} \boldsymbol{\mu}_A \\ \boldsymbol{\mu}_P \end{bmatrix} \\ &= \frac{1}{2}(\boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P) - \boldsymbol{q}_A^{\mathrm{T}}\boldsymbol{\mu}_A - \boldsymbol{q}_P^{\mathrm{T}}\boldsymbol{\mu}_P \\ &= \frac{1}{2}\boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PA}\boldsymbol{\mu}_A - \boldsymbol{q}_A^{\mathrm{T}}\boldsymbol{\mu}_A + constant. \end{aligned} \tag{B.137}$$

The equality constraint can be decomposed into $\mathbf{1}_A^{\mathrm{T}}\boldsymbol{\mu}_A = \nu - \mathbf{1}_P^{\mathrm{T}}\boldsymbol{\mu}_P$.

4.6 SMO Method for Hypersphere One-Class SVM

Let A only includes two working variables. We let $q_i = \frac{\nu}{2}k_{ii}$. The decomposition of the objective becomes

$$g(\mu_1, \mu_2) = \frac{1}{2}(k_{11}\mu_1^2 + 2k_{12}\mu_1\mu_2 + k_{22}\mu_2^2) + (\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{P1} - q_1)\mu_1 + (\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{P2} - q_2)\mu_2. \tag{B.138}$$

Let $v_i = \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{Pi} - q_i$, then the objective becomes

$$g(\mu_1, \mu_2) = \frac{1}{2}(k_{11}\mu_1^2 + 2k_{12}\mu_1\mu_2 + k_{22}\mu_2^2) + v_1\mu_1 + v_2\mu_2. \tag{B.139}$$

We do not want to compute v_i directly using $v_i = \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{Pi} - q_i$ because it takes linear time, and we have to store $\boldsymbol{K}$. Alternatively, we can compute it through the function $u_i' = f(\boldsymbol{x}_i)$ where the prime indicates the previous value. We know that

$$u_i' = R' - (k_{ii} - \frac{2}{\nu}\boldsymbol{K}_{i,:}\boldsymbol{\mu}' + \frac{1}{\nu^2}\boldsymbol{\mu}'^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu}') \tag{B.140}$$

Let $w' = \boldsymbol{\mu}'^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu}'$. Then

$$u_i' = R' - (k_{ii} - \frac{2}{\nu}(\boldsymbol{K}_{i,A}\boldsymbol{\mu}_A' + \boldsymbol{K}_{i,P}\boldsymbol{\mu}_P') + \frac{1}{\nu^2}w'). \tag{B.141}$$

Therefore

$$\begin{aligned}\boldsymbol{K}_{i,P}^{\mathrm{T}}\boldsymbol{\mu}'_P &= -\frac{\nu}{2}(R' - u'_i - k_{ii} - \frac{1}{\nu^2}w') - \boldsymbol{K}_{i,A}\boldsymbol{\mu}'_A \\ &= -\frac{\nu}{2}R' + \frac{\nu}{2}u'_i + \frac{\nu}{2}k_{ii} + \frac{1}{2\nu}w' - \boldsymbol{K}_{i,A}\boldsymbol{\mu}'_A. \end{aligned} \tag{B.142}$$

Therefore

$$\begin{aligned}v_i &= \boldsymbol{K}_{i,P}\boldsymbol{\mu}'_P - q_i \\ &= -\frac{\nu}{2}R' + \frac{\nu}{2}u'_i + \frac{\nu}{2}k_{ii} + \frac{1}{2\nu}w' - \boldsymbol{K}_{i,A}\boldsymbol{\mu}'_A - \frac{\nu}{2}k_{ii} \\ &= -\frac{\nu}{2}R' + \frac{\nu}{2}u'_i + \frac{1}{2\nu}w' - \boldsymbol{K}_{i,A}\boldsymbol{\mu}'_A. \end{aligned} \tag{B.143}$$

Therefore

$$v_1 = -\frac{\nu}{2}R' + \frac{\nu}{2}u'_1 + \frac{1}{2\nu}w' - k_{11}\mu'_1 - k_{12}\mu'_2, \tag{B.144}$$

$$v_2 = -\frac{\nu}{2}R' + \frac{\nu}{2}u'_2 + \frac{1}{2\nu}w' - k_{21}\mu'_2 - k_{22}\mu'_2. \tag{B.145}$$

Note that, if we alternatively define $s_i = \boldsymbol{K}_{i,:}\boldsymbol{\mu}'$ $(i = 1, \cdots, n)$ and keep updating it every iteration, v_i $(i = 1, 2)$ can be computed by

$$v_i = s'_i - k_{i1}\mu'_1 - k_{i2}\mu'_2 - q_i. \tag{B.146}$$

Also, we do not compute the constraint as $\mu_1 + \mu_2 = \nu - \mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}'_P$, because it takes linear time to compute $\mathbf{1}^{\mathrm{T}}\boldsymbol{\mu}'_P$. Alternatively, we use $\mu_1 + \mu_2 = \mu'_1 + \mu'_2 = t$.

Similar with the SMO for C-SVM, the second derivative along the linear line defined by $\mu_1 + \mu_2 = constant$ is $k_{11} + k_{22} - 2k_{12}$. If this value is non-negative then the two-variables problem has a minimum solution.

Solve Two-Variables Problem

Substituting $\mu_1 = t - \mu_2$ into the objective, we have

$$g(\mu_2) = \frac{1}{2}k_{11}(t - \mu_2)^2 + k_{12}(t - \mu_2)\mu_2 + \frac{1}{2}k_{22}\mu_2^2 + v_1(t - \mu_2) + v_2\mu_2. \tag{B.147}$$

Substituting $\mu_1 = t - \mu_2$ into the inequality constraints, we have

$$\begin{cases} 0 \leq t - \mu_2 \leq C \\ 0 \leq \mu_2 \leq C \end{cases}. \tag{B.148}$$

That is $L = \max(0, t - C)$ and $U = \min(C, t)$ where $C = \frac{1}{n}$. Taking the first derivative with respect to μ_2, and letting it to be zero, we have

$$\frac{\partial f(\mu_2)}{\partial \mu_2} = (k_{11} + k_{22} - 2k_{12})\mu_2 - k_{11}t + k_{12}t - v_1 + v_2 = 0. \tag{B.149}$$

Therefore

$$\begin{aligned} \mu_2 &= \frac{k_{11}t - k_{12}t + v_1 - v_2}{k_{11} + k_{22} - 2k_{12}} \\ &= \frac{k_{11}(\mu_1' + \mu_2') - k_{12}(\mu_1' + \mu_2') + \frac{\nu}{2}(u_1' - u_2') - k_{11}\mu_1' - k_{12}\mu_2' + k_{21}\mu_1' + k_{22}\mu_2'}{k_{11} + k_{22} - 2k_{12}} \\ &= \mu_2' + \frac{\frac{\nu}{2}(u_1' - u_2')}{k_{11} + k_{22} - 2k_{12}}. \end{aligned} \tag{B.150}$$

Alternatively, using s_1' and s_2', we can obtain the equivalent solution

$$\begin{aligned} \mu_2 &= \frac{k_{11}t - k_{12}t + v_1 - v_2}{(k_{11} + k_{22} - 2k_{12})} \\ &= \frac{k_{11}(\mu_1' + \mu_2') - k_{12}(\mu_1' + \mu_2') + s_1' - k_{11}\mu_1' - k_{12}\mu_2' - q_1 - s_2' + k_{21}\mu_1' + k_{22}\mu_2' + q_2}{k_{11} + k_{22} - 2k_{12}} \\ &= \mu_2' + \frac{s_1' - s_2' - q_1 + q_2}{k_{11} + k_{22} - 2k_{12}}. \end{aligned} \tag{B.151}$$

We then clip μ_2 into the bound $[L, U]$:

$$\mu_2 = \begin{cases} U & \text{if } \mu_2 \geq U \\ \mu_2 & \text{if } L \leq \mu_2 \leq U \\ L & \text{if } \mu_2 \leq L \end{cases}. \tag{B.152}$$

This formula is the update rule of μ_2. We can then update μ_1 through the constraint:

$$\mu_1 = t - \mu_2 = \mu_1' + \mu_2' - \mu_2. \tag{B.153}$$

Update s_i

$s_i = \boldsymbol{k}_{i,:}\boldsymbol{\mu}$ $(i = 1, \cdots, n)$ can be updated in constant time:

$$s_i = s'_i + k_{1i}(\mu_1 - \mu'_1) + k_{2i}(\mu_2 - \mu'_2). \tag{B.154}$$

Update w

$w = \boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu}$ can be decomposed into

$$\begin{aligned} w &= \boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P \\ &= k_{11}\mu_1^2 + k_{22}\mu_2^2 + 2k_{12}\mu_1\mu_2 + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{P,1}\mu_1 + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{P,2}\mu_2 + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P \\ &= k_{11}\mu_1^2 + k_{22}\mu_2^2 + 2k_{12}\mu_1\mu_2 + 2(s'_1 - k_{11}\mu'_1 - k_{21}\mu'_2)\mu_1 + 2(s'_2 - k_{12}\mu'_1 - k_{22}\mu'_2)\mu_2 \\ &+ \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P. \end{aligned} \tag{B.155}$$

We let

$$\begin{aligned} q &= w - w' \\ &= k_{11}(\mu_1^2 - {\mu'}_1^2) + k_{22}(\mu_2^2 - {\mu'}_2^2) + 2k_{12}(\mu_1\mu_2 - \mu'_1\mu'_2) \\ &+ 2(s'_1 - k_{11}\mu'_1 - k_{21}\mu'_2)(\mu_1 - \mu'_1) + 2(s'_2 - k_{12}\mu'_1 - k_{22}\mu'_2)(\mu_2 - \mu'_2), \end{aligned} \tag{B.156}$$

where $s'_i = \boldsymbol{k}_{i,:}\boldsymbol{\mu}'$ is cached. Alternatively, w can be updated by the following equation:

$$\begin{aligned} w &= \boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P \\ &= -\boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA} + 2\boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA}\boldsymbol{\mu}_A + 2\boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PA}\boldsymbol{\mu}_A + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P \\ &= -\boldsymbol{\mu}_A^{\mathrm{T}}\boldsymbol{K}_{AA} + 2s_1\mu_1 + 2s_2\mu_2 + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P \\ &= -k_{11}\mu_1^2 - k_{22}\mu_2^2 - 2k_{12}\mu_1\mu_2 + 2s_1\mu_1 + 2s_2\mu_2 + \boldsymbol{\mu}_P^{\mathrm{T}}\boldsymbol{K}_{PP}\boldsymbol{\mu}_P. \end{aligned} \tag{B.157}$$

We let

$$\begin{aligned} q &= w - w' \\ &= k_{11}({\mu'}_1^2 - \mu_1^2) + k_{22}({\mu'}_2^2 - \mu_2^2) + 2k_{12}(\mu'_1\mu'_2 - \mu_1\mu_2) + 2(s_1\mu_1 - s'_1\mu'_1) + 2(s_2\mu_2 - s'_2\mu'_2). \end{aligned} \tag{B.158}$$

Therefore

$$w = w' + q. \tag{B.159}$$

Update Center R

For $0 < \mu_i < C$ $(i = 1, 2)$, we have

$$R = k_{ii} - \frac{2}{\nu}\boldsymbol{k}_{i,:}\boldsymbol{\mu} + \frac{1}{\nu^2}\boldsymbol{\mu}^{\mathrm{T}}\boldsymbol{K}\boldsymbol{\mu}. \tag{B.160}$$

Now, we can update R:

$$R = k_{ii} - \frac{2}{\nu}s_i + \frac{1}{\nu^2}w. \tag{B.161}$$

Equivalently, we can update R by

$$\begin{aligned} R &= R' + \frac{2}{\nu}(s_i' - s_i) + \frac{1}{\nu^2}(w - w') \\ &= R' + \frac{2}{\nu}(k_{i1}(\mu_1' - \mu_1) + k_{i2}(\mu_2' - \mu_2)) + \frac{1}{\nu^2}q. \end{aligned} \tag{B.162}$$

If both μ_1 and μ_2 are at bound, then the situation is slightly complicated. We denote R_1, R_2 the radiuses computed according to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ respectively. We then can summarize all conditions in Table B.2.

Table B.2: Update R in various situations.

μ_1	μ_2	t	L	U	R
$0 < \mu_1 < C$	-	$0 < t < 2C$	-	-	R_1
-	$0 < \mu_2 < C$	$0 < t < 2C$	-	-	R_2
$0 < \mu_2 < C$	$0 < \mu_2 < C$	$0 < t < 2C$	-	-	$\frac{R_1+R_2}{2}$
0	0	0	0	0	$\max(R_1, R_2)$
0	C	C	0	C	$\frac{R_1+R_2}{2}$
C	0	C	0	C	$\frac{R_1+R_2}{2}$
C	C	2C	C	C	$\min(R_1, R_2)$

When $\mu_1 = \mu_2 = 0$ or $\mu_1 = \mu_2 = C$, it leads to $L = U$, therefore μ_1 and μ_2 equal to their previous values. Thus we can not update μ_1 and μ_2. These two situations need to be avoided earlier when selecting two variables.

Update u_i

u_i $(i = 1, \cdots, n)$ can be updated as below:

$$u_i = R - k_{ii} + \frac{2}{\nu}s_i - \frac{1}{\nu^2}w. \tag{B.163}$$

It can also been equivalently updated by

$$\begin{aligned} u_i &= u'_i + R - R' + \frac{2}{\nu}(s_i - s'_i) + \frac{1}{\nu^2}(w' - w) \\ &= u'_i + R - R' + \frac{2}{\nu}(k_{i1}(\mu_1 - \mu'_1) + k_{i2}(\mu_2 - \mu'_2)) - \frac{1}{\nu^2}q. \end{aligned} \tag{B.164}$$

Select Two Working Variables

The strategy of selecting two working variables is similar to that in the SMO for C-SVM, as described in Section 4.2.

Initialization

We can use the following method to initialize the algorithm by a feasible solution. We can first randomly select $s = \lfloor n\nu \rfloor$ Lagrange multipliers, and set them to $\frac{1}{n}$. Then, we set one multiplier to $\nu - \frac{s}{n}$. Finally, set the remaining multipliers to zero. Suppose $\nu = 0.55$, $n = 10$. We have $s = 5$, and $\nu - \frac{s}{n} = 0.05$. Therefore, we set five variables to 0.1, one variable to 0.05, and the remaining four variables to 0.

Bibliography

[1] M. Ochs and E. Fertig, "Matrix factorization for transcriptional regulatory network inference," in *CIBCB*, (Piscataway), pp. 387–396, IEEE CIS Society, IEEE Press, May 2012.

[2] R. Cho, M. Campbell, E. Winzeler, L. Steinmetz, A. Conway, L. Wodicka, T. Wolfsberg, A. Gareilian, D. Lockhart, and R. Davis, "A genome-wide transactional analysis of the mitotic cell cycle," *Molecular Cell*, vol. 2, pp. 65–73, July 1998.

[3] Y. Li and A. Ngom, "Sparse representation approaches for the classification of high-dimensional biological data," *BMC Systems Biology*, vol. 7, no. Suppl 4, p. S6, 2013.

[4] T. Furey, N. Cristianini, N. Duffy, D. Bednarski, M. Schummer, and D. Haussler, "Support vector machine classification and validation of cancer tissue samples using microarray expression data," *Bioinformatics*, vol. 16, no. 10, pp. 906–914, 2000.

[5] Y. Li and A. Ngom, "The regularized linear models and kernels toolbox in MATLAB," tech. rep., School of Computer Science, University of Windsor, Windsor, Ontario, 2013.

[6] M. Wall, A. Rechtsteiner, and L. Rocha, "Singular value decomposition and principal component analysis," in *A Practical Approach to Microarray Data Analysis* (D. Berrar, W. Dubitzky, and M. Granzow, eds.), pp. 91–109, Norwell, MA: Kluwer, 2003.

[7] C. Carvalho, J. Chang, J. Lucas, J. Nevins, Q. Wang, and M. West, "High-dimensional sparse factor modeling: applications in gene expression genomics," *Journal of the American Statistical Association*, vol. 103, no. 484, pp. 1438–1456, 2008.

[8] M. Elad, *Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing*. New York: Springer, 2010.

[9] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *Arxiv*, p. 1206.5538v2, 2012.

[10] M. Elad and M. Aharon, "Image denoising via learned dictionaries and sparse representation," in *CVPR*, (Washington DC), pp. 895–900, IEEE Computer Society, IEEE, June 2006.

[11] Y. Li and A. Ngom, "The non-negative matrix factorization toolbox for biological data mining," *BMC Source Code for Biology and Medicine*, vol. 8, no. 1, p. 10, 2013.

[12] D. D. Lee and S. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.

[13] D. Rowe, *Multivariate Bayesian Statistics: Models for Source Separation and Signal Unmixing*. Boca Raton, FL: CRC Press, 2003.

[14] M. West, "Bayesian factor regression models in the "large p, small n" paradigm," *Bayesian Statistics*, vol. 7, pp. 723–732, 2003.

[15] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From sparse solutions of systems of equations to sparse modeling of signals and images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, 2009.

[16] R. Tibshirani, "Regression shrinkage and selection via the Lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[17] Y. Li and A. Ngom, "Fast sparse representation approaches for the classification of high-dimensional biological data," in *IEEE International Conference on Bioinformatics and Biomedicine*, (Piscataway, NJ), pp. 306–311, IEEE, IEEE Press, Oct. 2012. acceptance rate 0.1993.

[18] Y. Li and A. Ngom, "Classification approach based on non-negative least squares," *Neurocomputing*, vol. 118, pp. 41–57, 2013.

[19] B. Olshausen and D. Field, "Sparse coding with an overcomplete basis set: a strategy employed by V1?," *Vision Research*, vol. 37, no. 23, pp. 3311–3325, 1997.

[20] X. Hang and F.-X. Wu, "Sparse representation for classification of tumors using gene expression data," *J. Biomedicine and Biotechnology*, vol. 2009, p. ID 403689, 2009.

[21] J. Wright, A. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation," *TPAMI*, vol. 31, no. 2, pp. 210–227, 2009.

[22] Y. Li, R. Caron, and A. Ngom, "A decomposition method for large-scale sparse coding in representation learning," in *World Congress on Computational Intelligence*, July 2014. To Be Submitted.

[23] S. J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior-point method for large-scale $l1$-regularized least squares," *IEEE J. Selected Topics in Signal Processing*, vol. 1, no. 4, pp. 606–617, 2007.

[24] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, UK: Cambridge University Press, 2004.

[25] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach, "Proximal methods for hierarchical sparse coding," *JMLR*, vol. 12, no. 2011, pp. 2297–2334, 2011.

[26] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, second ed., 2006.

[27] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Piladelphia: SIAM, 1995.

[28] T. Joachims, "Making large-scale support vector machine learning practical," in *Advances in Kernel Methods: Support Vector Learning* (B. Scholkopf, C. Burges, and A. Somla, eds.), ch. 11, pp. 169–184, MIT, 1998.

[29] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA: Athena Scientific, 2nd ed., 2008.

[30] J. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector Learning* (B. Scholkopf, C. Burges, and A. Somla, eds.), ch. 12, pp. 185–208, MIT, 1998.

[31] Z. Hu *et al.*, "The molecular portraits of breast tumors are conserved across microarray platforms," *BMC Genomics*, vol. 7, p. 96, 2006.

[32] E. I. Petricoin *et al.*, "Serum proteomic patterns for detection of prostate cancer," *J. National Cancer Institute*, vol. 94, no. 20, pp. 1576–1578, 2002.

[33] N. Dawany, W. Dampier, and A. Tozeren, "Large-scale integration of microarray data reveals genes and pathways common to multiple cancer types," *Int. J. Cancer*, vol. 128, no. 12, pp. 2881–2891, 2011.

[34] Y. Li and A. Ngom, "Fast kernel sparse representation approaches for classification," in *IEEE International Conference on Data Mining*, (Piscataway, NJ), pp. 966–971, IEEE, IEEE Press, Dec. 2012. acceptance rate 0.1997.

[35] C. Ding, T. Li, and M. I. Jordan, "Convex and semi-nonnegative matrix factorizations," *TPAMI*, vol. 32, no. 1, pp. 45–55, 2010.

[36] Y. Li and A. Ngom, "Versatile sparse matrix factorization and its application in high-dimensional biological data analysis," in *IAPR International Conference on Pattern Recognition in Bioinformatics*, pp. 91–101, Springer, June 2013.

[37] H. Zou and T. Hastie, "Regularization and variable selection via the elastic net," *Journal of the Royal Statistical Society - Series B: Statistical Methodology*, vol. 67, no. 2, pp. 301–320, 2005.

[38] H. Kim and H. Park, "Sparse non-negatice matrix factorization via alternating non-negativity-constrained least aquares for microarray data analysis," *SIAM J. Matrix Analysis and Applications*, vol. 23, no. 12, pp. 1495–1502, 2007.

[39] Y. Li and A. Ngom, "A new kernel non-negative matrix factorization and its application in microarray data analysis," in *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, (Piscataway, NJ), pp. 371–378, IEEE, IEEE Press, May 2012.

[40] D. Lee and S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, pp. 556–562, MIT Press, 2001.

[41] H. Kim and H. Park, "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method," *SIAM J. Matrix Analysis and Applications*, vol. 30, no. 2, pp. 713–730, 2008.

[42] C.-J. Lin, "On the convergence of multiplicative update algorithms for non-negative matrix factorization," *IEEE Transactions on Neural Networks*, vol. 18, pp. 1589–1596, 2007.

[43] Y. Li and A. Ngom, "Non-negative matrix and tensor factorization based classification of clinical microarray gene expression data," in *IEEE International Conference on Bioinformatics & Biomedicine*, (Piscataway, NJ), pp. 438–443, IEEE, IEEE Press, Dec. 2010.

[44] U. Alon *et al.*, "Broad patterns of gene expression revealed by clustering of tumor and normal colon tissues probed by oligonucleotide arrays," *PNAS*, vol. 96, no. 12, pp. 6745–6750, 1999.

[45] P. Kim and B. Tidor, "Subsystem identification through dimensionality reduction of large-scale gene expression data," *Genome Research*, vol. 13, pp. 1706–1718, 2003.

[46] M. Ochs, L. Rink, C. Tarn, S. Mburu, T. Taguchi, B. Eisenberg, and A. Godwin, "Detection of treatment-induced changes in signaling pathways in sastrointestinal stromal tumors using transcripttomic data," *Cancer Res.*, vol. 69, no. 23, pp. 9125–9132, 2009.

[47] Y. Li and A. Ngom, "Supervised dictionary learning via non-negative matrix factorization for classification," in *International Conference on Machine Learning and Applications*, (Piscataway, NJ), pp. 439–443, IEEE, IEEE Press, Dec. 2012.

[48] C.-H. Zheng, L. Zhang, T.-Y. Ng, S. Shiu, and D.-S. Huang, "Metasample-based sparse representation for tumor classification," *TCBB*, vol. 8, no. 5, pp. 1273–1282, 2011.

[49] Y. Li and A. Ngom, "Non-negative least squares for the classification of high dimensional biological data," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 10, no. 2, pp. 447–456, 2013.

[50] A. Blum and A. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, pp. 245–271, 1997.

[51] P. Mundra and J. Rajapakse, "Gene and sample selection for cancer classification with support vectors based t-statistic," *Neurocomputing*, vol. 73, no. 13-15, pp. 2353–2362, 2010.

[52] R. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, pp. 197–227, 1990.

[53] H. Liu, H. Motoda, and L. Yu, "A selective sampling approach to active feature selection," *Artificial Intelligence*, vol. 159, pp. 49–74, 2004.

[54] L. Bottou and V. Vapnik, "Local learning algorithms," *Neural Computation*, vol. 4, no. 6, pp. 888–900, 1992.

[55] V. Vapnik, "Principles of risk minimization for learning theory," in *Advances in Neural Information Processing Systems*, pp. 831–838, MIT Press, 1992.

[56] J. H. Friedman, "Local learning based on recursive covering," tech. rep., Department of Statistics, Sandford University, Standford, California, 1996.

[57] H. Cheng, P.-N. Tan, and R. Jin, "Efficient algorithm for localized support vector machine," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 4, pp. 537–549, 2010.

[58] C. Atkeson, A. Moore, and S. Schaal, "Locally weighted learning," *Artificial Intelligence Review*, vol. 11, pp. 11–73, 1997.

[59] A. Sierra and C. Cruz, "Global and local neural network ensembles," *Pattern Recognition Letters*, vol. 19, no. 8, pp. 651–655, 1998.

[60] J. Peng and B. Bhanu, "Local discriminative learning for pattern recognition," *Pattern Recognition*, vol. 34, pp. 139–150, 2001.

[61] V. Vapnik, *Statistical Learning Theory.* New York: Wiley-IEEE Press, 1998.

[62] O. Chapelle, B. Scholkopf, and A. Zien, *Semi-Supervised Learning.* Cambridge, MA: MIT Press, 2006.

[63] X. Zhu and A. Goldberg, *Introduction to Semi-Supervised Learning.* CA: Morgan & Claypool, 2009.

[64] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, no. 1-2, pp. 1–39, 2010.

[65] X. Li and H. Zhao, "Weighted random subspace method for high dimensional data classification," *Statistics and Its Interface*, vol. 2, pp. 153–159, 2009.

[66] R. Jacobs, I. Michael, S. Nowlan, and C. Hinton, "'adaptive mixtures of local experts," *Neural Computation*, vol. 3, pp. 79–87, 1991.

[67] S. Mukherjee, P. Tamayo, S. Rogers, R. Rifkin, A. Engle, C. Campbell, T. Golub, and J. Mesirov, "Estimating dataset size requirements for classifying DNA microarray data," *Journal of Computational Biology*, vol. 10, no. 2, pp. 119–142, 2003.

[68] C. Bi, M. Becker, and S. Leeder, "Derivation of minimum best sample size from microarray data sets: A Monte Carlo approach," in *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, (Piscataway, NJ), pp. 1–6, IEEE, IEEE Press, 2011.

[69] D. Notterman *et al.*, "Transcriptional gene expression profiles of colorectal adenoma, adenocarcinoma, and normal tissue examined by oligonucleotide arrays," *Cancer Research*, vol. 61, pp. 3124–3130, 2001.

[70] L. V. Veer *et al.*, "Gene expression profiling predicts clinical outcome of breast cancer," *Nature*, vol. 415, no. 6871, pp. 530–536, 2002.

[71] A. Rosenwald *et al.*, "The use of molecular profiling to predict survival after chemotherapy for diffuse large-B-cell lymphomar," *The New England Journal of Medicine*, vol. 346, no. 25, pp. 1937–1947, 2002.

[72] T. R. Golub *et al.*, "Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring," *Science*, vol. 286, no. 15, pp. 531–537, 1999.

[73] D. Beer *et al.*, "Gene-expression profiles predict survival of patientsene-expression profiles predict survival of patients with lung adenocarcinoma," *Nature Medicine*, vol. 8, no. 8, pp. 816–823, 2002.

[74] S. Pomeroy *et al.*, "Gene expression-based classification and outcome prediction of central nervous system embryonal tumors," *Nature*, vol. 415, no. 6870, pp. 436–442, 2002.

[75] J. Brunet, P. Tamayo, T. Golub, and J. Mesirov, "Metagenes and molecular pattern discovery using matrix factorization," *PNAS*, vol. 101, no. 12, pp. 4164–4169, 2004.

[76] D. Singh *et al.*, "Gene expression correlates of clinical prostate cancer behavior," *Cancer Cell*, vol. 1, no. 2, pp. 203–209, 2002.

[77] E. Yeoh *et al.*, "Classification, subtype discovery, and prediction of outcome in pediatric acute lymphoblastic leukemia by gene expression profiling," *Cancer Cell*, vol. 1, pp. 133–143, 2002.

[78] S. Pomeroy, P. Tamayo, M. Gaasenbeek, *et al.*, "Prediction of central nervous system embryonal tumour outcome based on gene expression," *Nature*, vol. 415, pp. 436–442, 2002.

[79] S. Armstrong *et al.*, "MLL translocations specify a distinct gene expression profile that distinguishes a unique leukemia," *Nature Genetics*, vol. 30, pp. 41–47, 2002.

[80] J. Khan *et al.*, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nature Medicine*, vol. 7, no. 6, pp. 673–679, 2001.

[81] N. Stransky *et al.*, "Regional copy number-independent deregulation of transcription in cancer," *Nature Genetics*, vol. 38, pp. 1386–1396, 2006.

[82] K. Chin *et al.*, "Genomic and transcriptional aberrations linked to breast cancer pathophysiologies," *Cancer Cell*, vol. 10, no. 6, pp. 529–541, 2006.

[83] J. Trolet *et al.*, "Genomic profiling and identification of high-risk uveal melanoma by array CGH analysis of primary tumors and liver metastases," *Invest Ophthalmol Vis Sci.*, vol. 50, no. 6, pp. 2572–2580, 2009.

[84] D. Chung and S. Keles, "Sparse partial least squares classification for high dimensional data," *Statistical Applications in Genetics and Molecular Bioinformatics*, vol. 9, no. 1, p. 17, 2010.

[85] J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[86] J. Driesen and H. V. hamme, "Modelling vocabulary acquisition, adaptation and generalization in infants using adaptive Bayesian PLSA," *Neurocmputing*, vol. 74, no. 11, pp. 1874–1882, 2011.

[87] P. Carmona-Saez, R. D. Pascual-Marqui, F. Tirado, J. M. Carazo, and A. Pascual-Montano, "Biclustering of gene expression data by non-smooth non-negative matrix factorization," *BMC Bioinformatics*, vol. 7, p. 78, 2006.

[88] G. Wang, A. Kossenkov, and M. Ochs, "LS-NMF: A modified non-negative matrix factorization algorithm utilizing uncertainty estimates," *BMC Bioinformatics*, vol. 7, p. 175, 2006.

[89] Y. Li and A. Ngom, "A new kernel non-negative matrix factorization and its application in microarray data analysis," in *CIBCB*, (Piscataway), pp. 371–378, IEEE CIS Society, IEEE Press, May 2012.

[90] A. Cichocki and R. Zdunek, "NMFLAB - MATLAB toolbox for non-negative matrix factorization," tech. rep., 2006.

[91] "The NMF: DTU toolbox," tech. rep., Technical University of Denmark.

[92] S. Liu, "NMFN: non-negative matrix factorization," tech. rep., Duke University, 2011.

[93] R. Gaujoux and C. Seoighe, "A flexible R package for nonnegative matrix factorization," *BMC Bioinformatics*, vol. 11, p. 367, 2010.

[94] Q. Qi, Y. Zhao, M. Li, and R. Simon, "Non-negative matrix factorization of gene expression profiles: A plug-in for BRB-ArrayTools," *Bioinformatics*, vol. 25, no. 4, pp. 545–547, 2009.

[95] E. Fertig, J. Ding, A. Favorov, G. Parmigiani, and M. Ochs, "CoGAPS: an R/C++ package to identify patterns and biological process activity in transcriptomic data," *Bioinformatics*, vol. 26, no. 21, pp. 2792–2793, 2010.

[96] S. Zhang, W. Wang, J. Ford, and F. Makedon, "Learning from incomplete ratings using non-negative matrix factorization," in *SDM*, (Philadelphia, PA), pp. 548–552, SIAM, 2006.

[97] C. Ding, T. Li, W. Peng, and H. Park, "Orthogonal nonnegative matrix tri-factorizations for clustering," in *KDD*, (New York), pp. 126–135, ACM, ACM, August 2006.

[98] R. Zass and A. Shashua, "Non-negative sparse PCA," in *NIPS*, NIPS, MIT Press, December 2006.

[99] N. Ho, *Nonnegative Matrix Factorization Algorithms and Applications*. PhD thesis, Department of Mathematical Engineering, Louvain-la-Neuve, Belgium, 2008.

[100] S. Madeira and A. Oliveira, "Biclustering algorithms for biological data analysis: A survey," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 1, no. 1, pp. 24–45, 2004.

[101] S. Draghici, P. Khatri, P. Bhavsar, A. Shah, S. Krawetz, and M. Tainsky, "Onto-Tools, the toolkit of the modern biologist: Onto-Express, Onto-Compare, Onto-Design and Onto-Translate," *Nucleic Acids Res.*, vol. 31, no. 13, pp. 3775–3781, 2003.

[102] H. Mewes, D. Frishman, C. Gruber, B. Geier, D. Haase, A. Kaps, K. Lemcke, G. Mannhaupt, F. Pfeiffer, C. Schuller, S. Stocker, and B. Weil, "MIPS: A database for genomes and protein sequences," *Nucleic Acids Res.*, vol. 28, no. 1, pp. 37–40, 2000.

[103] E. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J. Cherry, and G. Sherlock, "GO::TermFinder – open source software for accessing gene ontology information and finding significantly enriched gene ontology terms associated with a list of genes," *Bioinformatics*, vol. 20, pp. 3710–3715, 2004.

[104] D. Huang, B. Sherman, and R. Lempicki, "Systematic and integrative analysis of large gene lists using DAVID bioinformatics resources," *Nature Protoc.*, vol. 4, no. 1, pp. 44–57, 2009.

[105] D. Huang, B. Sherman, and R. Lempicki, "Bioinformatics enrichment tools: Paths toward the comprehensive functional analysis of large gene lists," *Nucleic Acids Res.*, vol. 37, no. 1, pp. 1–13, 2009.

[106] B. Tu, A. Kudlicki, M. Rowicka, and S. McKnight, "Logic of the yeast metabolic cycle: Temporal compartmentalization of cellular processes," *Science*, vol. 310, pp. 1152–1158, 2005.

[107] S. Chandriani, E. Frengen, V. Cowling, S. Pendergrass, C. Perou, M. Whitfield, and M. Cole, "A core myc gene expression signature is prominient in basal-like breast cancer but only partially overlaps the core serum response," *PloS ONE*, vol. 4, no. 5, p. e6693, 2009.

[108] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. Altman, "Missing value estimation methods for DNA microarrays," *Bioinformatics*, vol. 17, no. 6, pp. 520–525, 2001.

[109] I. Naseem, R. Togneri, and M. Bennamoun, "Linear regression for face recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 11, pp. 2106–2112, 2010.

[110] Y. Li and A. Ngom, "Classification of clinical gene-sample-time microarray expression data via tensor decomposition methods," in *2010 International Meeting on Computational Intelligence Methods for Bioinfomatics and Biostatistics*, vol. 6685, pp. 275–286, Springer, 2011.

[111] Y. Li and A. Ngom, "Mining gene-sample-time microarray data," in *Microarray Image and Data Analysis: Theory and Practice* (L. Rueda, ed.), CRC Press/Taylor & Francis, 2013. In Press.

[112] S. Baranzini, P. Mousavi, J. Rio, S. Caillier, A. Stillman, and P. Villoslada, "Transcription-based prediction of response to ifnβ using supervised computational methods," *PLOS Biology*, vol. 3, no. 1, p. e2, 2005.

[113] C. Perou, T. Sorlie, M. Eisen, *et al.*, "Molecular portraits of human breast tumours," *Nature*, vol. 406, pp. 747–752, 2000.

[114] B. Weinstock-Guttman, D. Badgett, K. Patrick, L. Hartrich, R. Santos, D. Hall, M. Baier, J. Feichter, and M. Ramanathan, "Genomic effects of ifn-β in multiple sclerosis patients," *The Journal of Immunology*, vol. 171, no. 5, pp. 2694–2702, 2003.

[115] T. Lin, N. Kaminski, and Z. Bar-Joseph, "Alignment and classification of time series gene expression in clinical studies," *Bioinformatics*, vol. 24, no. ISMB 2008, pp. i147–i155, 2008.

[116] I. Costa, A. Schonhuth, C. Hafemeister, and A. Schliep, "Constrained mixture estimation for analysis and robust classification of clinical time series," *Bioinformatics*, vol. 24, no. ISMB 2009, pp. i6–i14, 2009.

[117] L. Rueda and M. Herrera, "Linear dismensionality reduction by maximizing the Chernoff distance in the transformed space," *Pattern Recognition*, vol. 41, pp. 3138–3152, 2008.

[118] L. Omberg, G. Golub, and O. Alter, "A tensor higher-order singular value decomposition for integrative analysis of DNA microarray data from different studies," *Proceedings of the National Academy of Sciences*, vol. 104, no. 47, pp. 18371–18376, 2007.

[119] S. Ponnapalli, M. Saunders, C. V. Loan, and O. Alter, "A higher-order generalized singular value decomposition for comparison of global mrna expression from multiple organisms," *PLoS ONE*, vol. 6, no. 12, p. e28072, 2011.

[120] B. Savas, "Analyses and tests of handwritten digit recognition algorithms," Master's thesis, Dept. Mathathmatics Scientific Computing, Linköping University, Sweden, 2003.

[121] B. Savas and L. Elden, "Handwritten digit classification using higher order singular value decomposition," *Pattern Recongtion*, vol. 40, pp. 993–1003, 2007.

[122] C. Andersson and R. Bro, "The N-way toolbox for MATLAB," *Chemometrics and Intelligent Laboratory Systems*, vol. 52, pp. 1–4, 2000.

[123] M. Morup, L. Hansen, and S. Arnfred, "Algorithms for sparse nonnegative Tucker decompositions," *Neural Computation*, vol. 20, no. 8, pp. 2112–2131, 2008.

[124] K. Borgwardt, S. Vishwanathan, and H. Kriegel, "Class prediction from time series gene expression profiles using dynamical systems kernels," in *Pacific Symposium on Biocomputing*, pp. 547–558, World Scientific Press, 2006.

[125] T. Kolda and B. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.

[126] R. Edgar, M. Domrachev, and A. Lash, "Gene Expression Omnibus: NCBI gene expression and hybridization array data repository," *Nucleic Acids Res.*, vol. 30, no. 1, pp. 207–210, 2002.

[127] I. Guyon, J. Weston, and S. Barnhill, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, pp. 389–422, 2002.

[128] Y. Li, A. Ngom, and L. Rueda, "A framework of gene subset selection using multiobjective evolutionary algorithm," in *IAPR International Conference on Pattern Recognition in Bioinformatics*, pp. 38–48, Springer, 2012.

[129] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani, "1-norm support vector machines," in *NIPS*, vol. 16, NIPS, MIT Press, December 2003.

[130] Y. Li, B. Oommen, A. Ngom, and L. Rueda, "A new paradigm for pattern classification: nearest border techniques," in *26th Australasian Joint Conference on Artificial Intelligence*, pp. 441–446, Springer, Dec. 2013.

[131] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical pattern recognition: A review," *TAMI*, vol. 22, pp. 4 – 37, May 2000.

[132] B. V. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. Los Alamitos: IEEE Computer Society Press, 1991.

[133] J. C. Bezdek and L. I. Kuncheva, "Nearest prototype classifier designs: An experimental study," *International Journal of Intelligent Systems*, vol. 16, no. 12, pp. 1445 – 1473, 2001.

[134] S. Garcia, J. Derrac, J. R. Cano, and F. Herrera, "Prototype selection for nearest neighbor classification: Taxonomy and empirical study," *IEEE transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 417–435, 2012.

[135] I. Triguero, J. Derrac, S. Garcia, and F. Herrera, "A taxonomy and experimental study on prototype generation for nearest neighbor classification," *IEEE Trans. on Systems, Man and Cybernetics - Part C: App. and Re.*, vol. 42, pp. 86–100, 2012.

[136] P. E. Hart, "The condensed nearest neighbor rule," in *IEEE Transactions on Information Theory*, vol. 14, pp. 515–516, 1968.

[137] G. W. Gates, "The reduced nearest neighbor rule," in *IEEE Transactions on Information Theory*, vol. 18, pp. 431–433, 1972.

[138] C. L. Chang, "Finding prototypes for nearest neighbor classifiers," in *IEEE Transactions on Computing*, vol. 23, pp. 1179–1184, 1974.

[139] G. L. Ritter, H. B. Woodruff, S. R. Lowry, and T. L. Isenhour, "An algorithm for a selective nearest neighbor rule," in *IEEE Transactions on Information Theory*, vol. 21, pp. 665–669, 1975.

[140] I. Tomek, "Two modifcations of CNN," *IEEE Trans. Syst., Man and Cybern.*, vol. SMC-6, pp. 769 – 772, Nov. 1976.

[141] P. A. Devijver and J. Kittler, "On the edited nearest neighbor rule," in *Fifth International Conference on Pattern Recognition*, pp. 72–80, December 1980.

[142] K. Fukunaga, *Introduction to Statistical Pattern Recognition.* San Diego: Academic Press, second ed., 1990.

[143] Q. Xie, C. Laszlo, and R. K. Ward, "Vector quantization techniques for nonparametric classifier design," *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-15, pp. 1326 – 1330, Dec. 1993.

[144] T. Kohonen, *Self-Organizing Maps.* Springer, Berlin, 1995.

[145] S.-W. Kim and B. J. Oommen, "Enhancing prototype reduction schemes with lvq3-type algorithms," *Pattern Recognition*, vol. 36, pp. 1083–1093, May 2003.

[146] G. Li, N. Japkowicz, T. J. Stocki, and R. K. Ungar, "Full border identification for reduction of training sets," in *Canadian AI*, pp. 203–215, 2008.

[147] W. Duch, "Similarity based methods: a general framework for Classification, Approximation and Association," *Control and Cybernetics*, vol. 29, no. 4, pp. 937–968, 2000.

[148] G. M. Foody, "The significance of border training patterns in classification by a feed-forward neural network using back propogation learning," *International Journal of Remote Sensing*, vol. 20, no. 18, pp. 3549–3562, 1999.

[149] A. Thomas and B. J. Oommen, "The fundamental theory of optimal "anti-Bayesian" parametric pattern classification using order statistics criteria," *Pattern Recognition*, vol. 46, pp. 376–388, 2013.

[150] A. Thomas and B. J. Oommen, "Optimal order statistics-based "anti-Bayesian" parametric pattern classification for the exponential family," 2012. (To be submitted).

[151] A. Thomas and B. J. Oommen, "Order statistics-based parametric classification for multi-dimensional distributions," 2013. Submitted for Publication.

[152] D. Lee and J. Lee, "Domain described spport vector classifier for multi-classification problems," *Pattern Recognition*, vol. 40, pp. 41–51, 2007.

[153] T. Ban and S. Abe, "Implementing multi-class classifiers by one-class classification methods," in *IJCNN*, (Piscataway,NJ), pp. 327–332, IEEE, July 2006.

[154] D. Tax and R. Duin, "Support vector domain description," *Pattern Recognition Letters*, vol. 20, pp. 1191–1199, 1999.

[155] T. Mitchell, *Machine Learning*. Ohio: McGraw Hill, 1997.

[156] R. Tibshirani, T. Hastie, B. Narasimhan, and G. Chu, "Class prediction by nearest shrunken centroids, with applications to DNA microarrays," *Statistical Science*, vol. 18, pp. 104–117, Feb. 2003.

[157] B. Scholkopf, A. Smola, B. Williamson, and P. Bartlett, "New support vector algorithm," *Neural Computation*, vol. 12, pp. 1207–1245, 2000.

[158] D. Michie, D. Spiegelhalter, and C. Taylor, *Machine Learning, Neural and Statistical Classification*. New York: Ellis Horwood, 1994.

[159] K. Lee, J. Ho, and D. Kriegman, "Acquiring linear subspaces for dace recognition under variable lighting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 5, pp. 684–698, 2005.

[160] A. Frank and A. Asuncion, "UCI machine learning repository," tech. rep., University of California, Irvine, California, 2010.

[161] M. Maleki, M. Aziz, and L. Rueda, "Analysis of relevant physicochemical properties in obligate and non-obligate protein-protein interactions," in *4th IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pp. 345–351, IEEE, 2011.

[162] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," tech. rep., Department of Computer Science, National Taiwan University, Taipei,Taiwan, 2003.

[163] J. Hull, "A database for handwritten text recognition research," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 550–554, 1994.

[164] C.-C. Chang and C.-J. Lin, "Training ν-support vector classifiers: Theory and algorithms," *Neural Computation*, vol. 13, pp. 2119–2147, 2001.

[165] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011.

[166] I. Rezaeian, Y. Li, M. Crozier, E. Andrechek, A. Ngom, L. Rueda, and L. Porter, "Identifying informative genes for prediction of breast cancer subtypes," in *IAPR International Conference on Pattern Recognition in Bioinformatics*, pp. 138–148, Springer, June 2013.

[167] M. Diehn, "SOURCE: A unified genomic resource of functional annotations, ontologies, and gene expression data," *Nucleic Acids Research*, vol. 31, no. 1, pp. 219–223, 2003.

[168] Y. Li, N. Subhani, A. Ngom, and L. Rueda, "Alignment-based versus variation-based transformation methods for clustering microarray time-series data," in *ACM International Conference on Bioinformatics and Computational Biology*, (New York), pp. 53–61, ACM, ACM Press, Aug. 2010.

[169] M. Ramoni, P. Sebastiani, and I. Kohane, "Cluster analysis of gene expression dynamics," *PNAS*, vol. 99, pp. 9121–9126, June 2002.

[170] S. Tavazoie, J. Hughes, M. Campbell, R. Cho, and G. Church, "Systematic determination of genetic network architecture," *Nature Genetics*, vol. 22, pp. 281–285, 1999.

[171] P. Tamayo, D. Slonim, J. Mesirov, Q. Zhu, S. Kitareewan, E. Dmitrovsky, E. Lander, and T. Golub, "Interpreting patterns of gene expression with SOMs: Methods and application to hematopoietic differentiation," *PNAS*, vol. 96, pp. 2907–2912, March 1999.

[172] L. Bréhélin, "Clustering gene expression series with prior knowledge," *Lecture Notes in Computer Science*, vol. 3692, pp. 27–38, 2005.

[173] S. Chu, J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P. Brown, and I. Herskowitz, "The transcriptional program of sporulation in budding yeast," *Science*, vol. 282, pp. 699–705, 1998.

[174] L. Heyer, S. Kruglyak, and S. Yooseph, "Exploring expression data: Identification and analysis of coexpressed genes," *Genome Research*, vol. 9, pp. 1106–1115, 1999.

[175] J. Ernst, G. Nau, and Z. Bar-Joseph, "Clustering short time series gene expression data," *Bioinformatics*, vol. 21(suppl. 1), pp. i159–i168, 2005.

[176] Z. Bar-Joseph, G. Gerber, T. Jaakkola, D. Gifford, and I. Simon, "Continuous representations of time series gene expresion data," *Journal of Computational Biology*, vol. 10, pp. 341–356, July 2003.

[177] S. Djean, P. Martin, A. Baccini, and P. Besse, "Clustering time-series gene expression data using smoothing spline derivatives," *EURASIP Journal on Bioinformatics and Systems Biology*, vol. 70561, pp. 705–761, 2007.

[178] C. Moller-Levet, F. Klawonn, K. Cho, and O. Wolkenhauer, "Clustering of unevenly sampled gene expression time-series data," *Fuzzy sets and Systems*, vol. 152, pp. 49–66, May 2005.

[179] S. Peddada, E. Lobenhofer, L. Li, C. Afshari, C. Weinberg, and D. Umbach, "Gene selection and clustering for time-course and dose-response microarray experiments using order-restricted inference," *Bioinformatics*, vol. 19, no. 7, pp. 834–841, 2005.

[180] L. Rueda, A. Bari, and A. Ngom, "Clustering time-series gene expression data with unequal time intervals," *Springer Trans. on Compu. Systems Biology X*, vol. 10, no. 1974, pp. 100–123, 2008.

[181] N. Subhani, A. Ngom, L. Rueda, and C. Burden, "Microarray time-series data clustering via multiple alignment of gene expression profiles," *Springer Trans. on Pattern Recognition in Bioinformatics*, vol. 5780, pp. 377–390, 2009.

[182] J. H. Chiang, S. Yue, and Z. X. Yin, "A new fuzzy cover approach to clustering," *IEEE Transactions on Fuzzy Systems*, vol. 2, pp. 199–208, April 2004.

[183] Z. X. Yin and J. H. Chiang, "Novel algorithm for coexpression detection in time-varying microarry data sets," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 5, pp. 120–135, January/March 2008.

[184] N. Subhani, Y. Li, A. Ngom, and L. Rueda, "Alignment versus variation vector methods for clustering microarray time-series data," in *IEEE Congress on Evolutionary Computation*, (Piscataway, NJ), pp. 818–825, IEEE, IEEE Press, July 2010.

[185] R. Gâteaux, "Sur les fonctionnelles continues et les fonctionnelles analytiques," *Comptes rendus de l'academie des sciences (Paris)*, vol. 157, pp. 325–327, 1913.

[186] N. Subhani, L. Rueda, A. Ngom, and C. Burden, "Clustering microarray time-series data using expectation maximization and multiple profile alignment," in *IEEE International Conference on Bioinformatics and Biomedicine Workshops*, pp. 2–7, November 2009.

[187] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[188] R. Xu and D. Wunsch, *Clustering*. New Jersey: Wiley-IEEE Press, 2008.

[189] U. Maulik and S. Bandyopadhyay, "Performance evaluation of some clustering algorithms and validity indices," *IEEE Trans. Pattern Anal. Mach. Intell*, vol. 24, no. 12, pp. 1650–1654, 2002.

[190] X. Peng, R. K. M. Karuturi, L. D. Miller, K. Lin, Y. Jia, P. Kondu, L. Wang, L. S. Wong, E. T. Liu, M. K. Balasubramanian, and J. Liu, "Identification of cell cycle-regulated genes in fission yeast," *Molecular Biology of the Cell*, vol. 16, pp. 1026–1042, March 2005.

[191] H. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics*, vol. 52, no. 1, pp. 7–21, 2005.

[192] N. Subhani, L. Rueda, A. Ngom, and C. Burden, "Multiple gene expression profile alignment for microarray time-series data clustering," *Bioinformatics*, vol. 26, no. 18, pp. 2281–2288, 2010.

[193] S. Das, D. Caragea, S. M. Welch, and W. H. Hsu, *Handbook of Research on Computational Methodologies in Gene Regulatory Networks.* IGI Global, 2009.

[194] L. Chen, R.-S. Wang, and X-S.Zhang, *Biomolecular Networks: Methods and Applications in Systems Biology.* John Wiley & Sons, 2009.

[195] R. Daly, K. Edwards, J.S.O'Neill, S. Aitken, A. Millar, and M. Girolami, "Using higher-order dynamic Bayesian networks to model periodic data from the circadian clock of arabidopsis thaliana," *LNCS*, vol. 5780, pp. 67–78, 2009.

[196] I. Tsamardinos, L. Brown, and C. Aliferis, "The max-min hill-climbing Bayesian network structure learning algorithm," *Machine Learning*, vol. 65, pp. 31–78, 2006.

[197] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, "Using Bayesian networks to analyze expression data," *Journal of Computational Biology*, vol. 7, pp. 601–620, 2000.

[198] N. Friedman, K. Murphy, and S. Russell, "Learning the structure of dynamic probabilistic networks," in *Proc. the 14th Annual Conference on Uncertainty in Artificial Intelligence*, pp. 139–147, 1998.

[199] X. Xing and D. Wu, "Modeling multiple time units delayed gene regulatory network using dynamic Bayesian network," in *Proc. ICDMW*, (Hong Kong), pp. 190–195, IEEE, IEEE Press, 2006.

[200] J. Pearl, *Probabilistic Reasoning in Intelligent Systems.* San Mateo, CA: Morgan Kaufmann, 1988.

[201] D. Heckerman, D. Geiger, and D. Chickering, "Learning Bayesian networks: the combination of knowledge and statistical data," *Machine Learning*, vol. 20, pp. 197–243, 1995.

[202] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from imcomplete data via the EM algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977.

[203] K. Murphy, *Machine Learning: A Probabilistic Perspective.* Cambridge, MA: MIT Press, 2012.

[204] G. Schwarz, "Estimating the dimension of a model," *Ann. Stat.*, vol. 6, pp. 461–464, 1978.

[205] H. Chen, D. Maduranga, P. Mundra, and J. Zheng, "Integrating epigenetic prior in dynamic Bayesian network for gene regulatory network inference," in *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, (Piscataway, NJ), pp. 76–82, IEEE, IEEE Press, Apr. 2013.

[206] S. Imoto, T. Higuchi, T. Goto, K. Tashiro, S. Kuhara, and S. Miyano, "Combining microarrays and biological knowledge for estimating gene networks via Bayesian networks," *Journal of Bioinformatics and Computational Biology*, vol. 2, no. 1, pp. 77–98, 2004.

[207] R. Robinson, "Counting labeled acyclic digraphs," in *New directions in the theory of Graphs* (F. Harary, ed.), pp. 239–273, Academic Press, 1973.

[208] D. Chickering, "Learning Bayesian networks is NP-complete," in *Learning from Data: AI and Statistics V* (D. Frisher and H.-J. Lenz, eds.), Lecture Notes in Statistics, ch. 12, pp. 121–130, Springer, 1996.

[209] P. Spirtes, C. Glymour, and R. Scheines, *Causation, Prediction, and Search*. Cambridge, MA: MIT, second ed., 2001.

[210] Y. Li and A. Ngom, "The max-min high-order dynamic Bayesian network learning for identifying gene regulatory networks from time-series microarray data," in *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, (Piscataway, NJ), pp. 83–90, IEEE, IEEE Press, Apr. 2013.

[211] K. Murphy, *Dynamic Bayesian Networks:Representation, Inference and Learning*. PhD thesis, Berkley, CA, 2002.

[212] J. Rajapakse and I. Chaturvedi, "Gene regulatory networks with variable-order dynamic Bayesian networks," in *Proc. IJCNN*, (Barcelona, Spain), pp. 1–5, IEEE, IEEE Press, 2010.

[213] I. Chaturvedi and J. Rajapakse, "Building gene networks with time-delayed regulations," *Pattern Recognition Letters*, vol. 31, no. 14, pp. 2133 – 2137, 2010.

[214] D. Heckerman, "A tutorial on learning with Bayesian networks," in *Learning in Graphical Models* (M. Jordan, ed.), Adaptive Computation and Machine Learning series, ch. 11, pp. 301–354, Cambridge, MA: MIT, 1998.

[215] W. Buntine, "Theory refinement on Bayesian networks," in *Proc. UAI*, pp. 52–60, Morgan Kaufmann, 1991.

[216] G. Cooper and E. Herskovits, "A Bayesian method for the induction of probabilistic networks from data," *Machine Learning*, vol. 9, pp. 309–347, 1992.

[217] M. Zou and S. Conzen, "A new dynamic Bayesian network (dbn) approach for indentifying gene regulatory networks from time course microarray data," *Bioinformatics*, vol. 21, no. 1, pp. 71–79, 2005.

[218] D. Husmeier, "Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks," *Bioinformatics*, vol. 19, no. 17, pp. 2271–2282, 2003.

[219] Z. Ibrahim, A. Ngom, and A. Tawfik, "Using qualitative probability in reverse-engineering gene regulatory networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 2, pp. 326–334, 2011.

[220] M. Wellman, "Fundamental concepts of qualitative probabilistic networks," *Artificial Intelligence*, vol. 44, no. 3, pp. 257–303, 1990.

[221] M. Druzdzel and M. Henrion, "Efficient reasoning in qualitative probabilistic networks," in *AAAI*, pp. 548–553, AAAI, 1993.

[222] J. Yu, V. Smith, P. Wang, A. Hartemink, and E. Jarvis, "Advances to Bayesian network inference for generating causal networks from observational biological data," *Bioinformatics*, vol. 20, no. 18, pp. 3594–3603, 2004.

[223] I. Cantone, L. Marucci, F. Iorio, M. Ricci, V. Belcastro, M. Bansal, S. Santini, M. di Bernardo, D. di Bernardo, and M. Cosma, "A yeast synthetic network for in vivo assessment of reverse-engineering and modeling approaches," *Cell*, vol. 137, pp. 172–181, 2009.

[224] I. Simon, J. Barnett, N. Hannett, C. Harbison, N. Rinaldi, T. Volkert, J. Wyrick, J. Zeitlinger, D. Gifford, and T. Jaakkola, "Serial regulation of transcriptional regulators in the yeast cell cycle," *Cell*, vol. 106, pp. 697–708, 2001.

[225] P. Spellman, G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher, "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization," *Molecular Biology of The Cell*, vol. 9, pp. 3273–3297, 1998.

[226] H. Kim, G. H. Golub, and H. Park, "Missing value estimation for DNA microarray gene expression data: local least squares imputation," *Bioinformatics*, vol. 21, no. 2, pp. 187–198, 2005.

[227] T. Bo, B. Dysvik, and I. Jonassen, "LSimpute accurate estimation of missing values in microarray data with least squares methods," *Nucleic Acids Research*, vol. 32, no. 3, p. e34, 2004.

[228] D. Nguyen, N. Wang, and R. Carroll, "Evaluation of missing value estimation for microarray data," *Journal of Data Science*, vol. 2, pp. 347–370, 2004.

[229] G. Brock, J. Shaffer, R. Blakesley, M. Lotz, and G. Tseng, "Which missing value imputation method to use in expression profiles: a comparative study and two selection schemes," *BMC Bioinformatics*, vol. 9, p. 12, 2008.

[230] S. Oba, M. Sato, I. Takemasa, M. Monden, K. Matsubara, and S. Ishii, "Bayesian missing value estimation method for gene expression profile data," *Bioinformatics*, vol. 19, no. 16, pp. 2088–2096, 2003.

[231] Z. Bar-Joseph, G. Gerber, and D. Gifford, "Continuous representations of time-series gene expression data," *Journal of Computational Biology*, vol. 10, no. 3-4, pp. 341–356, 2003.

[232] M. Choong, M. Charbit, and H. Yan, "Autoregressive model based missing value estimation for DNA microarray time series data," *IEEE Trans. on Information Technology in Biomedicine*, vol. 13, no. 1, pp. 131–137, 2009.

[233] R. Bro, "PARAFAC. tutorial and applications," *Chemometric and Intelligent Laboratory Systems*, vol. 38, pp. 149–171, 1997.

[234] R. Bro, *Multi-way Analysis in the Food Industry: Models, Algorithms, and Applications*. PhD thesis, Department of Dairy and Food Science, Royal Veterinary and Agricultural University, Denmark, 1998.

[235] G. Tomasi and R. Bro, "PARAFAC and missing values," *Chemometric and Intelligent Laboratory Systems*, vol. 75, pp. 163–180, 2005.

[236] P. Paatero, "A weighted non-negative least squares algorithm for three-way PARAFAC factor analysis," *Chemometric and Intelligent Laboratory Systems*, vol. 38, pp. 223–242, 1997.

[237] R. Bro and S. D. Jong, "A fast non-negative constrained least squares algorithm," *Journal of Chemometrics*, vol. 11, pp. 393–401, 1997.

[238] R. Zhang, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Multi-category classification using extreme learning machine for microarray gene expression cancer diagnosis," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 3, pp. 485–495, 2007.

[239] Y. Li, A. Ngom, and L. Rueda, "Missing value imputation methods for gene-sample-time microarray data analysis," in *IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology*, (Piscataway, NJ), pp. 183–189, IEEE, IEEE Press, May 2010.

[240] T. Hastie, R. Tibshirani, G. Sherlock, M. Eisen, P. Brown, and D. Botstein, "Imputing missing data for gene expression arrays," tech. rep., Division of Biostatistics, Stanford University, 1999.

[241] L. Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.

[242] C. Andersson and R. Bro, "Improving the speed of multi-way algorithms: Part I. Tucker3," *Chemometrics and Intelligent Laboratory Systems*, vol. 42, pp. 93–103, 1998.

[243] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer, 2000.

[244] C. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[245] A. Hoerl, "Application of ridge analysis to regression problems," *Chemical Engineering Progress*, vol. 58, pp. 54–59, 1962.

[246] V. Vapnik, *Estimation of Dependences Based on Empirical Data*. New York: Springer-Verlag, 1982.

[247] C. Cortes and V. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[248] G.-X. Yuan, K.-W. Chang, C.-J. Hsieh, and C.-J. Lin, "A comparison of optimization methods and software for large-scale l1-regularized linear classification," *Journal of Machine Learning Research*, vol. 11, pp. 3183–3234, 2010.

[249] G.-X. Yuan, C.-H. Ho, and C.-J. Lin, "Recent advances of large-scale linear classification," *Proceedings of the IEEE*, vol. 100, no. 9, pp. 2584–2603, 2012.

[250] H. Drucker, C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *NIPS*, vol. 9, pp. 155–161, MIT Press, 1996.

[251] C.-C. Chang and C.-J. Lin, "Training ν-support vector regression: Theory and algorithms," *Neural Computation*, vol. 14, no. 8, pp. 1959–1977, 2002.

[252] B. Scholkopf, J. Platt, J. Shawe-Taylor, A. Smola, and B. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Computation*, vol. 13, pp. 1443–1471, 2001.

[253] P. Mundra and J. Rajapakse, "SVM-RFE with MRMR filter for gene selection," *IEEE Transactions on Nanobioscience*, vol. 9, no. 1, pp. 31–37, 2010.

[254] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *The Fifth Annual Workshop on Computational Learning Theory*, (New York), pp. 144–152, ACM, 1992.

[255] E. Osuna, R. Freund, and F. Girosi, "An improved training algorithm for support vector machines," in *Neural Networks for Signal Processing [1997] VII. Proceedings of the 1997 IEEE Workshop*, (Piscataway,NJ), pp. 276–285, IEEE, September 1997.

[256] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," Tech. Rep. MSR-TR-98-14, Microsoft Research, 1998.

[257] R.-E. Fan, P.-H. Chen, and C.-J. Lin, "Working set selection using second order information for training support vector machines," *JMLR*, vol. 6, pp. 1889–1918, 2005.

[258] P.-H. Chen, C.-J. Lin, and B. Scholkopf, "A tutorial on ν-support vector machines," *Applied Stochastic Models in Business and Industy*, vol. 21, pp. 111–136, 2005.

Vita Auctoris

Yifeng Li received his Bachelor's and Master's Degrees from Shandong Institute of Light Industry (now Qilu University of Technology), China, in 2006, and 2009, respectively. He studies in the School of Computer Science, University of Windsor, Canada, from 2009 to 2013 for a Degree of Doctoral Philosophy. His research interests include sparse representation and probabilistic graphical models in machine learning, high dimensional data analysis and systems biology in bioinformatics, and numerical optimization.