12-2012

Learning understandable classifier models.

Jan Chorowski
University of Louisville

LEARNING UNDERSTANDABLE CLASSIFIER MODELS

By

Jan Chorowski
MS, Wroclaw University of Technology, 2009

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering
University of Louisville
Louisville, Kentucky

December 2012

LEARNING UNDERSTANDABLE CLASSIFIER MODELS

By

Jan Chorowski
MS, Wroclaw Technical University, 2009

A Dissertation Approved On

November 14, 2012
Date

by the following Dissertation Committee:

Jacek M. Zurada, Dissertation Director

Tamer Inanc

Mehmed Kantardzic

Karla Conn Welch

Olfa Nasraoui

ACKNOWLEDGEMENTS

I would first like to thank my advisor, Dr. Jacek M. Zurada, Director of the Computational Intelligence Laboratory at the University of Louisville, for his guidance, encouragement, and patience. Dr. Zurada has been more than an academic advisor to me, often offering guidance and help in many a, whether big or petty, matter that concerned my life in Louisville. I would also like to thank Dr. Tamer Inanc, Dr. Mehmed Kantardzic, Dr. Karla Conn Welch, and Dr. Olfa Nasraoui for agreeing to serve on the dissertation committee. I am grateful for your encouragement and advice.

I am also deeply indebted to my friends and colleagues Artur and Jordan for helping me to survive in this new place and to Tomasz, Sera, Chase, and Karina for rendering my life here colorful and enjoyable. I am grateful to my collaborators Jian, Tolga, Ehsan, and Qinwei, for many fruitful discussions. I also thank the wonderful people from the Catholic Campus Ministry for helping me to grow spiritually. My deepest appreciation is due to Lisa Bell and other administrative assistants at the University of Louisville for helping me to navigate among formal requirements.

Most importantly I thank my wife, Agata, for her patience and constant support while this dissertation has slowly been taking its current form. I am grateful for the encouragement, advice, and support that my parents, Małgorzata and Maciej, gave me. Equally supportive were my brother Jakub, his wife Urszula, and my sister Maria, as well as my grand-parents Maria, Janina, Stanisław, and Bohdan, who did not live to see the completion of this degree.

Finally, I thank the God the Creator who has been ultimately responsible for every success in this process.

ABSTRACT

LEARNING UNDERSTANDABLE CLASSIFIER MODELS

Jan Chorowski

November 14, 2012

The topic of this dissertation is the automation of the process of extracting understandable patterns and rules from data. An unprecedented amount of data is available to anyone with a computer connected to the Internet. The disciplines of Data Mining and Machine Learning have emerged over the last two decades to face this challenge. This has led to the development of many tools and methods. These tools often produce models that make very accurate predictions about previously unseen data. However, models built by the most accurate methods are usually hard to understand or interpret by humans. In consequence, they deliver only decisions, and are short of any explanations. Hence they do not directly lead to the acquisition of new knowledge. This dissertation contributes to bridging the gap between the accurate opaque models and those less accurate but more transparent for humans.

This dissertation first defines the problem of learning from data. It surveys the state-of-the-art methods for supervised learning of both understandable and opaque models from data, as well as unsupervised methods that detect features present in the data. It describes popular methods of rule extraction from unintelligible models which rewrite them into an understandable form. Limitations of rule extraction are described. A novel definition of understandability which ties computational complexity and learning is provided to show that rule extraction is an NP-hard problem. Next, a discussion whether one can expect that even an accurate classifier has learned new knowledge. The survey ends with a presentation of two approaches to building of understandable classifiers. On the one hand, understandable models must be able to accurately describe relations in the data. On the other hand, often a description of the output of a system in terms of its input requires the introduction of intermediate concepts, called features. Therefore it is crucial to develop methods that describe the data with understandable features and are able to use those features to present the relation that describes the data.

Novel contributions of this thesis follow the survey. Two families of rule extraction algorithms are considered. First, a method that can work with any opaque classifier is introduced. Artificial training patterns are generated in a mathematically sound way and used to train more accurate understandable models. Subsequently, two novel algorithms that require that the opaque model is a Neural Network are presented. They rely on access to the network's weights and biases to induce rules encoded as Decision Diagrams.

Finally, the topic of feature extraction is considered. The impact on imposing non-negativity constraints on the weights of a neural network is considered. It is proved that a three layer network with non-negative weights can shatter any given set of points and experiments are conducted to asses the accuracy and interpretability of such networks. Then, a novel path-following algorithm that finds robust sparse encodings of data is presented.

In summary, this dissertation contributes to improved understandability of classifiers in several tangible and original ways. It introduces three distinct aspects of achieving this goal: infusion of additional patterns from the underlying pattern distribution into rule learners, the derivation of decision diagrams from neural networks, and achieving sparse coding with neural networks with non-negative weights.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Automatic data acquisition, storage, and processing is easier and more ubiquitous nowadays than ever. An important emerging goal is to automatically learn new knowledge out of the vast and constantly growing amount of available data. The disciplines of Data Mining (DM) and Machine Learning (ML) provide necessary algorithms and tools to detect patterns of interest and relations present in the data. However, the comprehensive, if not the ultimate, goal of this query is not to just detect these relations, but to transform them into new knowledge expressed in a form possibly readily understandable and usable by humans. Ideally, the results produced by DM and ML methods should not only provide numerical predictions, but also enable humans to understand patterns and regularities present in the data whenever possible and applicable.

The notion of understandability is inherently subjective. Moreover, it is variable even for a single individual. Usually, the understandability increases with the time spent on the analysis of a problem. This suggests that the derivation of knowledge from data is a continuous process rather than a one-time application of one or more generic methods. Successful approaches must allow for cooperation and synergy between the data analyst and the tools used. Ultimately, the analyst should be able to inject known facts and relations about the data into the learning process.

This leads to an important problem of how to enhance or extend an existing body of knowledge. Most often, the problem-at-hand is not entirely unknown. The most common and generic way is to develop a proper problem description. The data mining process will be most successful if relevant features are used to represent the data. Moreover, a judicious choice of features may help circumvent known deficiencies of selected methods. For instance, decision trees and production rules can partition the input space using only axis-parallel planes. However, some methods allow for the inclusion of more background information about a problem. Often information about problem constraints or invariants can be included in the classifier's design. This topic is continued in Chapter II, Section 3.

There are many cases of successful application of machine learning methods to obtain new knowledge. A classical one is the derivation of rules predicting soybean diseases [1]. The rules produced by the AQ11 program outperformed those written by an expert. Other examples are poisonous mushroom detection [2] or protein secondary structure prediction [3].

Despite these examples, the problem of automatically eliciting knowledge from data is far from being solved. According to a recent study [4], the best supervised learning methods are Boosted [5] and Bagged [6] Decision Trees, Random Forests [7], Support Vector Machines [8], and Neural Networks [9]. Each of these methods produce results which are inherently hard to understand. Ensembles of decision trees are unintelligible because one has to analyze large amount of trees in entirety. Equally hard to understand are Neural Networks and Support Vector Machines because they express their models in the form of complicated mathematical formulas. As a result, the most accurate methods can only be used as black-boxes: even though they provide the best results, all the knowledge is hidden. This precludes the usage of such opaque classifiers in many application domains, which require both justifications of decisions and understandable conclusions from machines. For instance, the Equal Credit Opportunity Act requires that an explanation can be provided for credit refusal. Similarly, a medical diagnosis has to be preferably based on premises that the physician thoroughly understands.

The following four chapters of this dissertation cover the state-of-the art in learning from data, describe methods of rewriting opaque models into rules, and discuss algorithms to find characteristic features of the data. Chapter II begins with a description of the theoretical foundations of learning from data. It surveys the state-of-the art methods that produce both opaque and understandable models. It describes methods for unsupervised data description using features. It then surveys the most important rule extraction methods. It describes limitations of the rule extraction process and discusses whether one can expect that even an accurate classifier has learned new knowledge. Chapter II ends with the presentation of two approaches to building of understandable classifiers. First, the data needs to be described using meaningful concepts, called features. Then, those features can be used to describe the relations present in the data.

Chapters III and IV concentrate on improving the accuracy of understandable models that use the original data description. Chapter III presents mathematically sound methods of generating additional training patterns on which interpretable models of increased accuracy can be induced. Chapter IV presents two novel methods of rewriting Neural Networks into Decision Diagrams, a data structure that is especially suitable for rule extraction.

Chapter V discusses the problem of extracting meaningful features from data. A novel method of imposing non-negativity constraints on weights of Artificial Neural Networks to learn understandable and discriminative features is presented and discussed. The chapter is concluded with a discussion of the problem of robust sparse coding. A novel algorithm is presented that computes the changes of the encoding when sparsity level is varied. The proposed method has running times favorable to traditional linear programming approaches.

CHAPTER II

LEARNING UNDERSTANDABLE MODELS

This chapter begins with a description of the problem of learning classifiers from data. Then, it describes popular architectures that often learn accurate, but incomprehensible models. Next, it surveys the state-of-the art methods that directly learn human-readable models. Popular unsupervised feature detection methods are described using a common encoder-decoder framework. Then, rule extraction methods which induce an understandable model by using a given accurate, but incomprehensible one are presented. Subsequently, limitations of rule extraction are discussed. A definition of understandability is introduced to prove computational infeasibility of exact rule extraction. Finally, the question of how much information can be obtained from a given accurate black-box classifier is discussed.

A Fundamentals of Learning From Data

Supervised learning of understandable classifier models is a subfield of the more general problem of learning from data. The Statistical Learning Theory is the leading theory that defines and analyzes the problem of learning form limited amounts of data. In this section the main assumptions of the Statistical Learning Theory are presented. Finally, the Probably Approximately Correct theory is described because its formulation is more intuitive in a rule extraction context.

The Statistical Learning Theory (SLT), also known as the Vapnik-Chervonenkis (VC) theory is considered to be the best currently available theory for flexible statistical estimation with finite samples [10]. It formally states the problem of learning from data and provides a methodology for learning. It provides bounds on the performance of the learning process when only a finite

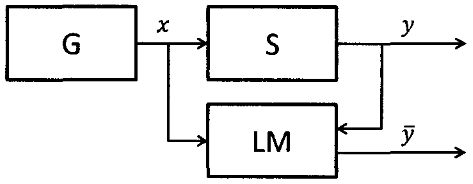


Figure 1: The elements of a learning system. After Vapnik [11].

amount of samples is available. The SLT's model of learning from labeled examples contains three elements that are pictured in Figure 1 [11]:

1. The generator of the data (examples), G.
2. The target operator (also called supervisor's operator or supervisor), S.
3. The learning machine, LM.

The generator G samples the vectors $\boldsymbol{x}$ independently and identically distributed (i.i.d.) according to an unknown, but fixed probability distribution $P(\boldsymbol{x})$. The supervisor processes the input vectors $\boldsymbol{x}$ into outputs y by sampling from a conditional distribution $P(y|\boldsymbol{x})$ (this includes the case of a deterministic output function $y = f(\boldsymbol{x})$). When the supervisor performs pattern recognition the vector $\boldsymbol{x}$ contains the attributes of a sample and the output y is a discrete class label. Again, the distribution $P(y|\boldsymbol{x})$ is unknown, but fixed. The generator and supervisor thus generate a training set of pairs sampled independently and identically from $P(y, \boldsymbol{x}) = P(y|\boldsymbol{x})P(\boldsymbol{x})$.

The learning machine LM is capable of implementing a set of functions $f(\boldsymbol{x}, \alpha), \alpha \in \Lambda$. It observes a finite amount of training samples $(y_i, \boldsymbol{x}_i), i = 1, \ldots, N$ (the training set) composed of the system's outputs y and inputs $\boldsymbol{x}$. It is then tasked with choosing a function from the given set that will best approximate the supervisor's output.

The Statistical Learning Theory distinguishes between two goals for the learning machine [11]:

- To *imitate* the supervisor's operator by constructing an operator that provides for a given generator G the best prediction to the supervisor's outputs.
- To *identify* the supervisor's operator by constructing an operator which is close

to the supervisor's operator.

Classical statistical methods first choose a family of probability distributions that match the problem. Then the parameters of the distribution are optimized to match the data by maximizing the likelihood. Thus they lean toward system identification. In contrast to classical parametric statistics, the SLT concentrates on the task of system imitation, which is easier than identification [11].

The remaining question is which mechanism shall the learning machine use to best approximate the supervisor's operation using but a limited number of training samples. First a suitable measure of discrepancy between supervisor's and learning machine's outputs must be specified. In SLT this discrepancy is measured with a *loss function* $L(y, f(\boldsymbol{x}, \alpha))$. For the pattern recognition problem the loss function may be an error indicator [11]:

$$L(y, f(\boldsymbol{x}, \alpha)) = \begin{cases} 0 & \text{if } y = f(\boldsymbol{x}, \alpha) \\ 1 & \text{otherwise.} \end{cases} \tag{1}$$

The learning machine should ideally minimize the expected value of the loss over the distribution of data. This quantity has been named in SLT the *risk functional* [12]:

$$R(\alpha) = \int L\left(y, f(\boldsymbol{x}, \alpha)\right) dP(y, \boldsymbol{x}). \tag{2}$$

The risk functional cannot be directly minimized because it depends on the unknown probability distribution $P(y, \boldsymbol{x})$. However, an estimate of the loss may be computed using the training data. Define the *empirical risk* to be [12]:

$$R_{emp}(\alpha) = \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, f(\boldsymbol{x}_i, \alpha)\right) \tag{3}$$

The SLT proposes to minimize the empirical risk and provides conditions under which the true risk will also be minimized. This is called the Empirical Risk Minimization (ERM) principle and consists of approximating the minimum of the true risk functional (2) by finding the minimum of the empirical risk (3).

One of the main results of SLT is a bound of the true risk expressed as a function of the empirical risk, number of training samples, and the learning machine's Vapnik-Chervonenkis (VC) dimension. The VC dimension h of a learning machine LM is defined to be the maximum number of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_h$ which can be separated in all 2^h possible ways (*shattered*) by the LM. Intuitively, the VC dimension grows with the amount of concepts or pattern that the LM can discern. The relation bounding the risk is as follows [12]. Assume that the loss functional is bounded:

$$0 \leq L(y, f(\boldsymbol{x}, \alpha)) \leq B, \qquad \alpha \in \Lambda. \tag{4}$$

Then with probability at least $1 - \eta$, the inequality

$$R(\alpha) \leq R_{emp}(\alpha) + \frac{B\epsilon}{2}\left(1 + \sqrt{1 + \frac{4R_{emp}(\alpha)}{B\epsilon}}\right) \tag{5}$$

holds true simultaneously for all function of the set (4), where:

$$\epsilon = 4\frac{h\left(\ln\frac{2N}{h} + 1\right) - \ln\eta}{N}, \tag{6}$$

where h is the VC dimension. In particular, this bound holds for the function $f(\boldsymbol{x}, \alpha_0)$ which minimizes the empirical risk [12].

The bound (5) is too loose to be practically useful in predicting classifier performance. However, its main implication is that model complexity has to be chosen as a function of the number of available samples. This forms the basis of the Structural Risk Minimization principle. It also suggests, that the accuracy of a learning machine can be improved by using more training data, which needs to be sampled form the unknown probability distribution $P(y, \boldsymbol{x})$.

An illustration of this property is presented in Figure 2. A Random Forest, C4.5 decision tree and RIPPER production rules were trained using Weka [7,13–16] on the "Yeast" data from the UCI repository [17]. The figure shows an average of 10 runs of 10-fold stratified cross-validation in which the algorithm is allowed to use only a fraction of the training data. It can be seen that for all three classifiers the accuracy improves with the addition of training data. This observation can form the basis of a simple, yet effective rule extraction scheme that is investigated in Chapter III.

The Probably Approximately Correct (PAC) theory [18] of learning looks at the problem of learning from a different perspective. It allows the learning machine

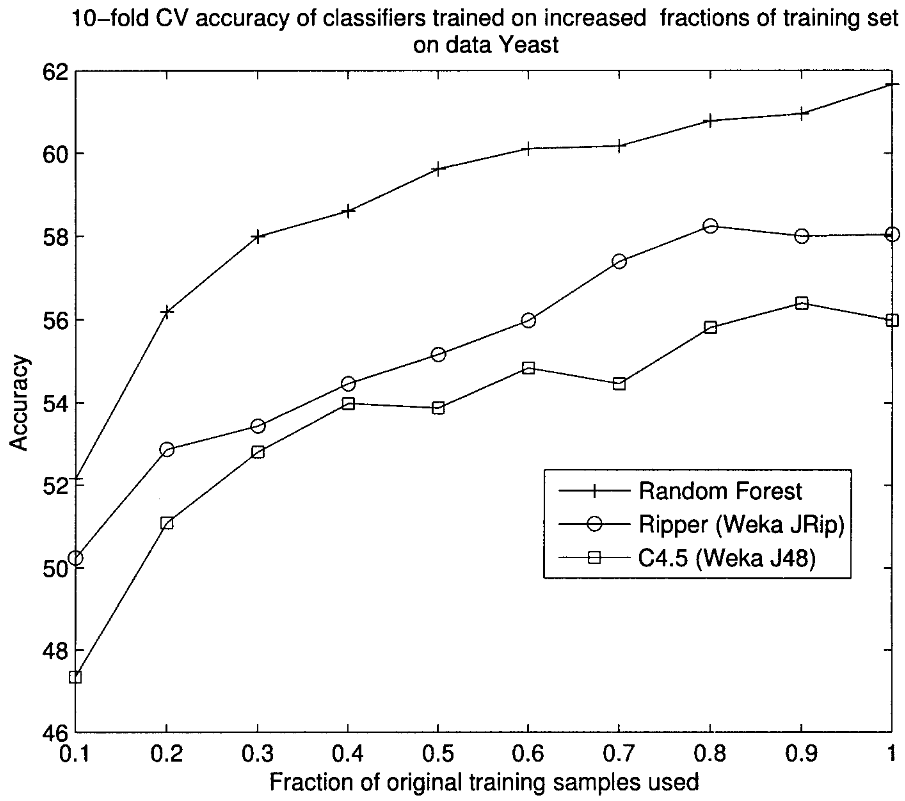


Figure 2: Increasing 10xCV accuracy as more data is used for training.

to consult an *examples* function that generates data and an *oracle* function that classifies the data. However, the learner must perform only a polynomial amount of steps. This definition ties the problem of learning from data to the computational complexity of algorithms. Similarly to the SLT, PAC provides bounds on the accuracy of a learner with respect to the number of examples queried and size of the hypothesis space. In fact, PAC learning can be unified with SLT [19].

Just like the SLT, the PAC model relies on the assumption that there is an unknown, but fixed probability distribution of the data $P(y, \boldsymbol{x})$. The learning machine LM has access to two functions: *examples()* which generates new i.i.d. samples from the distribution $P(\boldsymbol{x})$ and *oracle($\boldsymbol{x}$)* that returns the class of the vector $\boldsymbol{x}$. A concept class F is defined to be learnable if there exist an algorithm A that [18]:

1. Runs in time polynomial in an adjustable parameter η, in the various parameters that quantify the size of the classifier to be learned, and in the number of dimensions of $\boldsymbol{x}$, n.

2. For all concepts $f \in F$ and all distributions P over data, the algorithm will deduce with probability at least $(1 - \eta^{-1})$ a classifier $g \in F$ that makes errors with probability at most η^{-1}.

The rule extraction problem consists of finding an understandable representation of a given black-box classifier. If this black-box classifier is treated as the *oracle*, then the PAC learning model can provide both an inspiration for rule extraction methods and a better understanding of rule extraction methods.

B Popular Black-box Models

Many popular classifiers produce models that are difficult to understand. This section introduces the main classifiers that belong to this category: Feedforward Neural Networks, Support Vector Machines, and Ensemble Methods. Main assumptions that underlie their operation are discussed, along with reasons why their understandability is often very poor.

The description of the black-box methods will rely on the following matrix notation. Without loss of generality, suppose the problem is described using k real-valued attributes and that the labels belong to a finite set $\mathcal{C}$. The training set is composed of N pairs $(\boldsymbol{x}_i, y_i), i = 1, \ldots, N$ of input attribute vectors $\boldsymbol{x}_i \in \mathbb{R}^k$ and discrete class labels $y_i \in \mathcal{C}$.

1 Feedforward Neural Networks

Feedforward Neural Networks implement a nonlinear function $f : \mathbb{R}^k \to \mathcal{C}$ that typically maps the input patterns $\boldsymbol{x}$ into the set $\mathcal{C}$ of class labels [9]. Let the labels be consecutive integers: $\mathcal{C} = \{1, 2, \ldots, m\}$. The network's function f is parameterized by matrices of weights and biases. Often the output of the network is a real vector of size $m = |\mathcal{C}|$ that indicates for every class label the probability that the processed sample belongs to the given class. The network is trained by changing the weights and biases to minimize a loss function that measures the discrepancy between network predictions and the known class label.

The l-th layer of a Neural Network maps a d_{in} dimensional input vector to a d_{out} dimensional output one. It is parameterized by a matrix of weights $\boldsymbol{W}_l \in \mathbb{R}^{d_{out} \times d_{in}}$ and by a bias vector $\boldsymbol{b}_l \in \mathbb{R}^{d_{out}}$. For an input vector $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$ the vector of layer activation values $\boldsymbol{a}_l \in \mathbb{R}^{d_{out}}$ is defined to be:

$$\boldsymbol{a}_l = \boldsymbol{W}_l \boldsymbol{x} + \boldsymbol{b}_l. \tag{7}$$

The layer's output vector $\boldsymbol{o}_l$ is formed by applying a *transfer function* to the vector of neuron activations $\boldsymbol{a}_l$.

Often the transfer function is a univariate nonlinear sigmoidal (shaped like the letter "S") one. Commonly used are the logistic sigmoid $\sigma(x) = \frac{1}{1+\exp(-x)}$ and the hyperbolic tangent $\sigma(x) = \tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$. The layers' output is formed by an element-wise application of the function $\sigma(\cdot)$ to the vector of activations, which is denoted:

$$\boldsymbol{o}_l = \sigma(\boldsymbol{a}_l). \tag{8}$$

Another popular transfer function, used mostly for the last layer is the *SoftMax* function that transforms a vector of arbitrary real numbers into a vector of numbers from the range $(0, 1)$ that moreover sum to 1 and hence can be interpreted as probabilities. The j-th element of the output of the SoftMax function is defined by:

$$\text{SoftMax}(\boldsymbol{x})[j] = \frac{\exp(\boldsymbol{x}[j])}{\sum_{i=1}^{I} \exp(\boldsymbol{x}[i])}, \tag{9}$$

where $\boldsymbol{x}[j]$ denotes the j-th element of $\boldsymbol{x}$. Unlike the sigmoid functions that operate on single elements, the SoftMax function transforms a whole vector of neuron activations into a vector of outputs.

A multilayer Feedforward Neural Network is obtained by composing L individual layers:

$$\begin{aligned} \boldsymbol{o}_1(\boldsymbol{x}) &= \sigma_1(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) \\ \boldsymbol{o}_l(\boldsymbol{x}) &= \sigma_l(\boldsymbol{W}_l \boldsymbol{o}_{l-1} + \boldsymbol{b}_l) \text{ for } l = 2, \ldots, L. \end{aligned} \tag{10}$$

Loss functions are used to measure the discrepancy between $\boldsymbol{o}_L(\boldsymbol{x})$, the output of the network for a sample $\boldsymbol{x}$, and the desired class label y. A commonly chosen measure is the sum-of-squares loss:

$$Loss(\boldsymbol{o}_L(\boldsymbol{x}), y) = \sum_{c \in \mathcal{C}} \left(\boldsymbol{o}_L(\boldsymbol{x})[c] - \mathcal{I}\{y = c\}\right)^2, \tag{11}$$

where $\mathcal{I}\{\cdot\}$ denotes the indicator function, which takes the value 1 if the statement inside of the brackets is true and 0 otherwise.

Alternatively, if the outputs of the network are interpreted as the conditional probability distribution of the class labels given the attributes, the loss may be formulated as the negative log-likelihood of observing a given sample $(\boldsymbol{x}, y)$:

$$Loss(\boldsymbol{o}_L(\boldsymbol{x}), y) = \sum_{c \in \mathcal{C}} \mathcal{I}\{y = c\} \log\left(\frac{\mathcal{I}\{y = c\}}{\boldsymbol{o}_L(\boldsymbol{x})[c]}\right) = -\log\left(\boldsymbol{o}_L(\boldsymbol{x})[y]\right). \tag{12}$$

Network training consists of changing the weights and biases to minimize the sum of the loss function computed on all training samples and of penalties incurred by weights and biases. The penalties serve to *regularize* the network by e.g. preferring small values of the weights. The total target minimized during network training is:

$$T = \frac{1}{N}\sum_{1}^{N} Loss(\boldsymbol{o}_L(\boldsymbol{x}_i), y_i) + \sum_{l=1}^{L}\left(Pw(\boldsymbol{W}_l) + Pb(\boldsymbol{b}_l)\right), \tag{13}$$

where Pw ad Pb denote the penalties imposed on weights and biases. Networks are often trained by minimizing the target T using a first order gradient minimization. The new weights $\boldsymbol{W}'$ and biases $\boldsymbol{b}'$ are computed from their old values ($\boldsymbol{W}$, $\boldsymbol{b}$) by performing a single step along the gradient:

$$\begin{aligned} \boldsymbol{W}_l' &= \boldsymbol{W}_l - \eta\nabla_{\boldsymbol{W}_l}T \text{ for } l = 1, \ldots, L \\ \boldsymbol{b}_l' &= \boldsymbol{b}_l - \eta\nabla_{\boldsymbol{b}_l}T, \text{ for } l = 1, \ldots, L, \end{aligned} \tag{14}$$

where η is the learning rate that controls the length of the gradient step. The training is often accelerated by using only a small fraction of training samples is used to compute the target T in (13) and its gradient for the weights and biases update (14). In this way training consists of many small updates that use a noisy gradient estimate. For this reason this training regime is often called *stochastic gradient descent* [20].

On the other hand, second order methods may be used to speed the convergence of training. The methods that employ them usually need to process the whole training set and are often referred to as *batch methods.* Especially popular are the Scaled Conjugate Gradient [21] and the Levenberg-Marquardt method [22].

Feedforward Neural Networks are difficult to understand because of the large number of weights and biases that define them, and because of the inherent nonlinearity exhibited by their operation. As such, they were the first classifiers for which special methods were devised to better understand their meaning [23]. On the one hand, *pruning methods* that are described next try to increase network understandability by simplifying its architecture. On the other hand,*rule extraction* methods, that are described in Section E, try to rewrite a given network into a set of comprehensible rules.

Pruning methods were designed to simplify networks by removing spurious units and connections inside a network [24]. Such methods as OBD [25] and OBS [26] prune a trained network using second-order derivative information to estimate the impact of removing a connection. An algorithm that removes hidden nodes and adjusts remaining weights by solving a system of equations is presented in [27]. Other pruning methods augment the loss criterion minimized during network training by adding terms that promote the reduction of the number of connections or by adding terms that enforce other constraints that simplify the network. Weight decay is the mechanism traditionally used to reduce the magnitude of network weights by penalizing the sum of their squares. Enhanced sparsity of weights can be enforced by penalizing instead the sum of weights' absolute values [28]. This mechanism is similar to the elastic net feature selection technique used in linear regression [29]. Often particular values of network weights are required. Soft weight sharing [30] aims at clustering weight values. A polynomial penalty is used in [2] to constrain the weights to be zero or ± 1. Hyperbolic tangent nonlinearity has been applied to weights for the same purpose in [31]. Use of those techniques also facilitates enhanced understanding of the network because the analysis of interactions between signals incoming to a neuron is greatly simplified if the weights amplifying those signals are similar for all inputs.

2 Support Vector Machines

The Support Vector Machine uses two main ideas. First, kernel functions are used to transform the problem from the original input space into a highly dimensional one, called the feature space, where linear separation of training samples belonging to different classes is possible. Second, to find the best separating hyperplane, the concept of maximum margin is introduced. Finally, the optimization problem which defines the SVM is convex and quadratic, and therefore it can be solved efficiently [8, 32–35].

The Support Vector Machine was originally proposed for problems with two classes only. For ease of notation the two class labels will be ± 1, i.e. $\mathcal{C} = \{-1, 1\}$. It will be assumed that the function $\phi(\cdot) : \mathbb{R}^k \to \mathbb{R}^p$ is given. It maps a given sample $\boldsymbol{x} \in \mathbb{R}^k$ into a p-dimensional space, that will be called the *feature* space. The function ϕ is usually specified through the use of kernel functions, that are defined next. The SVM operates by finding a linear boundary that separates samples in the feature space. The *margin* of the separating hyperplane is defined to be the smallest distance from the hyperplane to a training sample. The wider the margin the smaller is the impact of small perturbations of the decision boundary to classifier performance. Therefore a wide-margin classifier will have less tendency to over-fit the data and will yield a better testing accuracy [35]. The SVM will now be formally defined. The linear decision boundary in the feature space corresponds to:

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \phi(\boldsymbol{x}) + b, \tag{15}$$

where $\boldsymbol{w}$ is a weight vector and b is a bias term (offset form the origin). It can be shown [35, 36] that the margin is inversely proportional to $\boldsymbol{w}^t\boldsymbol{w}/2$. Thus the separating hyperplane that maximizes the margin is found by solving:

$$\begin{gathered} \min_{\boldsymbol{w},b} \frac{\boldsymbol{w}^T\boldsymbol{w}}{2} \\ \text{subject to: } (\boldsymbol{w}^T\phi(\boldsymbol{x}_i) + b)y_i \geq 1 \text{ for } i = 1, \ldots, N. \end{gathered} \tag{16}$$

The constraints ensure that all training samples are correctly separated by the hyperplane, while the optimization target maximizes the margin.

Sometimes it is necessary to let the classifier do a few errors on the training set. If training samples are allowed into the margin region, the margin can be larger. In this way better performance on the testing set is obtained, at the price of lowering the performance on the training set. The soft-margin SVM can be defined by the following optimization problem [8, 33–35]:

$$\begin{aligned} &\min_{\boldsymbol{w},b,\epsilon} \frac{\boldsymbol{w}^T\boldsymbol{w}}{2} + C\sum_{i=1}^{N}\epsilon_i \\ &\text{subject to: } (\boldsymbol{w}^T\phi(\boldsymbol{x}_i)+b)y_i \geq 1-\epsilon_i \text{ for } i=1,\ldots,N \\ &\epsilon_i \geq 0 \text{ for } i=1,\ldots,N, \end{aligned} \tag{17}$$

where ϵ_i denotes the margin violation of the i-th training sample (which is 0 for samples outside of the margin region) and the constant C determines the trade-off between the margin width and the sum of margin violations.

The problem (17) is an instance of the convex quadratic programming problem and it can be solved by finding points that satisfy the Karush-Kuhn-Tucker (KKT) optimality conditions [37]. Due to the inequality constraints the solution itself is a quadratic programming problem [8, 33–35]:

$$\begin{aligned} &\max_{\alpha} -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) + \sum_{i=1}^{N}\alpha_i \\ &\text{subject to: } \sum_{i=1}^{N} \boldsymbol{y}_i\alpha_i = 0 \\ &0 \leq \alpha_i \leq C \text{ for } i=1,\ldots,N, \end{aligned} \tag{18}$$

where α_i are the Lagrange multipliers. This formulation of the SVM problem is notable for two reasons. First, the samples in the feature space are accessed only through their inner products, i.e. $\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j)$. Second, the weights defining the decision boundary are not present and need to be recovered from the multipliers α_i as:

$$\boldsymbol{w} = \sum_{i=1}^{N}\alpha_i\phi(\boldsymbol{x}_i). \tag{19}$$

Moreover, most of the coefficients α_i are zero [8]. Only the samples with nonzero multipliers contribute to the decision boundary. For this reason they are called the Support Vectors.

The SVM depends on inner products between samples only, which makes it possible to use kernel functions K that define the transformation ϕ implicitly. Formally, if a function $K : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$ operates on pairs of samples in the attribute domain and satisfies Mercer's condition, then there exists a space in which K defines an inner product operation:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j). \tag{20}$$

Thus it is possible to use a kernel function that will transform the training samples into a highly (possibly infinitely) dimensional space in which a linear separating boundary can be found, without ever needing to compute the mapping of samples into the feature space. In practice, the Gaussian kernel, parameterized by the constant γ is frequently used for SVM training:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma ||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2). \tag{21}$$

The SVM is usually trained by minimizing (18) using the Sequential Minimal Optimization (SMO) algorithm [38]. The SMO algorithm operates by repeatedly selecting two multipliers α and maximizing the target of (18) with respect to those two multipliers only. This is a simple problem, because the two multipliers are tied through the equality constraint. Thus at every step, the SMO algorithm minimizes a univariate quadratic function, for which the analytical solution exists.

New samples in a nonlinear SVM are classified using the relation:

$$\boldsymbol{w}^T \phi(\boldsymbol{x}) + b = \sum_{i=1}^{N} \alpha_i \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}) + b = \sum_{i=1}^{N} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b, \tag{22}$$

where the equation (19) was used to determine the weights based on multipliers α. The decision boundary of the SVM is difficult to understand because it involves a weighted contribution of many support vectors. Also, due to the nonlinear nature of kernel functions, the exact impact of a support vector on a given sample is often difficult to assess.

3 Ensemble Methods

The ensemble methods aggregate many simple classifiers that are called *base learners* into a single one (the ensemble). The main motivation is that if the base learners make errors independently, then the errors cancel when aggregated, just like random noise variance diminishes when measurements are averaged. Breiman suggested that the ensemble's accuracy depends on two traits of base learners: their average strength and the mean correlation between them [7]:

$$GeneralizationError \leq \frac{\rho(1-s^2)}{s^2}, \tag{23}$$

where ρ is the mean value of correlation between base learners and s is the strength defined to be the mean value of the margin of the base learners. The margin is defined to be the mean value of the difference between the probability assigned by the base learner to the correct class, and the highest probability that the base learner assigns to an incorrect class. The relation (23) suggests that an ensemble will be successful when the base learners are at the same time strong and not correlated.

In practice, decision trees are often used as base learners because they are fast to train and often highly accurate. Moreover, decision trees are unstable, because they are sensitive to small changes in the learning conditions. Ensemble methods often vary the data on which individual trees are induced to amplify the instability of decision trees. In consequence, the correlations between trees are lowered.

In the Bagging method [39], *bootstrap* samples are used to train the base learners. A bootstrap sample of the training data is formed by sampling from it uniformly and with replacement. A bootstrap sample of the same size as the original training set contains about 2/3 of its unique samples. Different base learners are thus trained on different training sets. Moreover, the samples that do not enter a given bootstrap sample may be used to assess the quality of each base learner.

In the Bagging ensemble each base learner is assigned a single vote. To classify new data each base learner casts its vote according to its prediction. The ensemble then tallies individual votes to select its own prediction. The ratio of votes cast for different classes can be used to compute the probabilities assigned by the ensemble to class labels.

Another method of varying the training data consists of describing the training samples with a random selection of attributes [40]. It has been named *the Random Subspace method*, because each base learner is trained on a projection of the entire training data onto the space spanned by a few randomly selected attributes.

The Bagging and Random Subspace approaches are combined in the Random Forest (RF) classifier [7]. The RF uses decision trees as the base learners. However, during tree induction node splits are selected by considering only a few randomly selected attributes. Moreover, each random tree is trained on a bootstrap sample of the dataset. The randomization introduced during tree induction slightly weakens individual trees. However, it also decorrelates them. Overall accuracy of the ensemble is increased. Moreover, individual tree induction in a Random Forest is fast because only a fraction of attributes is considered to choose each split.

The Boosting ensemble building method follows a different path [5]. In contrast to Bagging and Random Forests, base learners must be trained sequentially. This is because the training set used to induce a new base learner depends on the accuracy of the base learners that have already been included into the ensemble.

The AdaBoost boosting algorithm adds base learners to the ensemble one-by-one [5]. Each base learner is induced on a sample of the training set drawn according to training instance weights, w. After the base learner is constructed its accuracy is computed on the original, unweighted, training set and used to assign a coefficient α to the base learner. The training set weights w are then modified to reflect the errors made by the base learner. Then a new set is drawn to train the next base learner.

The Boosted ensemble classifies new data by computing a weighted average of the votes cast by base learners with weights corresponding to their coefficients, α. In this way inaccurate base learners affect the decision less than the accurate ones.

There are many explanations of the good accuracy of the boosting approach. Its inventors, Shapire, Freud, et al., demonstrate that boosting operates by increasing the margin of the ensemble [41]. In similar spirit, Rosset shows how choosing the base learner coefficients α corresponds to a coordinate descent optimization of a margin-maximizing function [42]. On the other hand, Breiman, the inventor of Random Forests, conjunctures that in its final stages AdaBoost essentially emulates a Random Forest [7].

Ensemble methods produce results that are difficult to understand because they are often composed of dozens or even hundreds of base learners. The answers of the base learners are averaged, which precludes their individual analysis. Instead, combinations of attributes that cause the majority of base learners to act in a specified way must be sought for, which reduces to analyzing the exponentially many different combinations of individual base learner outputs.

C Popular White-box Models

Methods that produce directly readable results are usually called *white-boxes.* They typically express their models in the form of decision trees or production rules, however, other formats such as decision tables [43] or decision diagrams with exceptions [44] have been also proposed.

Production rules are expressions of the form "if *conditions* then *classification*". Algorithms directly inducing them usually employ the "separate and conquer" approach in which rules are added one-by-one to an initially empty set. After a new rule is added to the rule set, training samples covered by it are put aside (or separated, justifying the name of the methodology) and a new rule maximizing some criterion on the remaining samples is sought for. A final pruning step can be used to further simplify and ameliorate the produced rule set. The search for a single rule can proceed in several different ways:

General-to-specific search starts with an empty conjunction of tests (matching everything) to which new tests are added until the rule matches samples from one class only. This approach is used in the CN2 [45], RIPPER [14], PART [46] methods, and AQ family of methods (which, however, seeds the rules with a positive example) [1].

Specific-to-general search starts with a rule covering just one training sample.

Conditions are dropped (the rule is generalized) until it covers enough samples while still being pure. This strategy is used by the rule extraction method described in Chapter IV, Section B.

Other search methods, such as training a perceptron with constrained weights and rewriting it as rules which is used in the MLP2LN [2] method.

The process of sequential covering of the rule space can be analyzed and visualized using ROC diagrams [47].

Decision Tree learners, such as C4.5 [13] or CART [48], greedily build a tree whose nodes contain tests on attributes, directed edges point to test outcomes, and leaves represent the predicted class. Usually the tests depend on just one attribute, which is selected to maximize a measure of tree purity. Typically, the tree purity is measured by entropy of class distribution at a node or the GINI statistic. To increase their understandability, decision trees can be converted into production rules. In fact, every path from the root to a leaf forms one rule. However, for better understandability further simplification and processing steps which reduce the rule set are required [13]. Decision trees are very popular and many implementations exist, notably in the publicly available Weka [15] data mining suite.

Most white-box supervised learning methods accept input in the usual tabular format of data in which every training instance is described using a fixed set of attributes. A notable exception is the FOIL program, which directly processes logical relations [49]. It should be noted that decision tree and production rule learners work by selecting the best test on data from a specified set. They can be extended to support other descriptions of data by specifying specialized tests. A further discussion of this topic is presented by Breiman et al. [48].

D Feature Detection Techniques

It is estimated that 60% of the effort spent on the data mining process is devoted to preparation and understanding of the data [50]. Clearly, it is important to properly select and define the attributes that will be used to describe the data. The design of good attributes is also a natural place to introducing prior knowledge and beliefs into the data mining process. The techniques of feature detection are designed to help in this task. They often produce descriptions of the data and distill the information present in the data to reduce the burden placed on subsequent analysis steps.

The feature detection problem is described as follows. Let the training set contain pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, N$ of input attribute vectors $\boldsymbol{x}_i \in \mathbb{R}^k$ and class labels y_i. The features of the sample $(\boldsymbol{x}_i, y_i)$ are defined to be a vector $\boldsymbol{v}_i \in \mathbb{R}^p$ which can be computed using the input alone $\boldsymbol{v}_i = f_e(\boldsymbol{x}_i)$. The features cannot depend on the class of a sample, because such dependency would preclude their computation on unlabeled test samples. Usually the features are detected in an unsupervised manner, i.e. without using the labels. Therefore features are often designed to preserve most of the information present in the original data, but in a simpler way. For instance feature vectors $\boldsymbol{v}$ often have a lower dimensionality than the attribute vectors $\boldsymbol{x}$.

Many unsupervised learning techniques can be described with a unified energy-based framework [51–53]. An energy function that assigns energy values to pairs of attributes $\boldsymbol{x}_i$ and features $\boldsymbol{v}_i$ is defined. Learning of energy-based models consists of finding configurations of method parameters, attributes, and features that have low energy values. Inference is executed as a minimization of the energy of a trained model over either the attributes, or the features. The following discussion of unsupervised learning algorithms is a simplification of the energy-based approach to their analysis.

Feature detection methods often depend on linear algebra techniques. For a matrix $\boldsymbol{M}$ let $\boldsymbol{M}_{:i}$ denote the i-th column of $\boldsymbol{M}$ and let $\boldsymbol{M}_{j:}$ denote the j-th row.The matrix of data attributes $\boldsymbol{X}$ is formed by horizontal concatenation of attribute vectors. Every column of $\boldsymbol{X}$ contains the attributes of a data sample $\boldsymbol{X}_{:i} = \boldsymbol{x}_i$. Likewise, let $\boldsymbol{V}$ denote the matrix of data features with $\boldsymbol{V}_{:i} = \boldsymbol{v}_i$.

Usability of a feature set mandates the existence of the *encoder* function, f_e. The encoder induces a matching *decoder* function f_d that reconstructs the attribute vector from the features. Decoding a vector of features is the task of finding an attribute vector whose encoding will match the given vector of features:

$$f_d(\boldsymbol{v}) = \arg\min_{\boldsymbol{x}} ||\boldsymbol{v} - f_e(\boldsymbol{x})||, \tag{24}$$

where $||\cdot||$ denotes a suitable measure of discrepancy between $\boldsymbol{v}$ and $f_e(\boldsymbol{x})$. It is often the Euclidean norm, which results in a least squares minimization problem.

Some feature extraction schemes begin with the definition of the decoder. The encoding of an input vector $\boldsymbol{x}$ is then specified implicitly in a manner similar to (24), as the vector of features that, when decoded, provides a good reconstruction of the input $\boldsymbol{x}$:

$$f_e(\boldsymbol{x}) = \arg\min_{\boldsymbol{v}} ||\boldsymbol{x} - f_d(\boldsymbol{v})||. \tag{25}$$

1 Methods Using Linear Encoders

Many popular feature detection and dimensionality reduction techniques define the features to be a linear projection of the input data:

$$f_e(\boldsymbol{x}) = \boldsymbol{M}_E\boldsymbol{x}. \tag{26}$$

Using matrix notation the feature matrix is obtained by matrix multiplication of the attribute matrix: $\boldsymbol{V} = \boldsymbol{M}_E\boldsymbol{X}$. Since the encoder is explicitly specified, a matching decoder has to be found from (24). Methods with linear encoders often use the Euclidean norm in the decoder. Then (24) becomes a least squares problem, with closed-form solution given by the Moore-Penrose pseudo-inverse of $\boldsymbol{M}_E$ [54]. Moreover, if the rows of $\boldsymbol{M}_E$ are orthonormal the pseudo-inverse is just $\boldsymbol{M}_E^T$: $\boldsymbol{M}_E^\dagger = \boldsymbol{M}_E^T(\boldsymbol{M}_E\boldsymbol{M}_E^T)^{-1} = \boldsymbol{M}_E\boldsymbol{I}^{-1} = \boldsymbol{M}_E^T$.

The Principal Components Analysis (PCA) is obtained when variance of the features is maximized [35]. It is computed by performing an eigendecomposition of the data covariance matrix [54]. If $cov(\boldsymbol{X}) = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, then the encoding matrix is $\boldsymbol{M} = \boldsymbol{Q}^T$. Moreover, since $\boldsymbol{Q}$ is an unitary matrix the data can be reconstructed from features by $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{V}$. For dimensionality reduction only the p eigenvectors that correspond to the p largest eigenvalues are used to form the encoding matrix.

The Independent Component Analysis (ICA) looks for an encoding matrix $\boldsymbol{M}_E$ such that the features $\boldsymbol{v}$ are statistically independent for a given sample $\boldsymbol{x}$. In practice, the features are found by maximizing the non-Gaussianity of the projected data [55].

The Singular Value Decomposition (SVD) can be used to decompose the attribute matrix $\boldsymbol{X}$ into a product of two unitary matrices and a diagonal matrix of singular values: $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ [54]. Let $\boldsymbol{U}_p$, $\boldsymbol{\Sigma}_p$, $\boldsymbol{V}_p$ be the submatrices of $\boldsymbol{U}$, $\boldsymbol{\Sigma}$, and $\boldsymbol{V}$, respectively, that correspond to the p largest singular values. Then $\boldsymbol{X}_p = \boldsymbol{U}_p\boldsymbol{\Sigma}_p\boldsymbol{V}_p^T$ is a rank p approximation of $\boldsymbol{X}$ with the lowest sum of squares residual error. Encoding matrix $\boldsymbol{M}_E$ is computed as $\boldsymbol{M}_E = \boldsymbol{\Sigma}_p^{-1}\boldsymbol{U}_p$.

2 Methods Using Linear Decoders

Often the feature vectors $\boldsymbol{v}$ must have some special properties. For example they must be sparse or non-negative. In those cases it easier to specify how the attributes are reconstructed from the features. Subsequently, the encoder is defined with a constrained optimization problem (28). The downside is that most often there exist no closed-form solutions of (28) that can be used to compute the feature vectors for the data.

In the most general form, the decoding operation is defined using a decoding matrix $\boldsymbol{M}_D$:

$$\boldsymbol{x} = f_d(\boldsymbol{v}) = \boldsymbol{M}_D\boldsymbol{v}. \tag{27}$$

Moreover, the feature vectors must belong to the set $\mathcal{C}$ of feature vectors that satisfy the constraints of this particular method.

Finding the encoding of an input vector $\boldsymbol{x}_i$ now requires to solve a constrained optimization problem:

$$\begin{aligned} \boldsymbol{v}_i = f_e(\boldsymbol{x}_i) = \arg\min_{\boldsymbol{v}} \|\boldsymbol{x} - \boldsymbol{M}_D\boldsymbol{v}\| \\ \text{subject to: } \boldsymbol{v} \in \mathcal{C} \end{aligned} \tag{28}$$

The matrix $\boldsymbol{M}_D$ is often determined from the data in a process called feature detection. Usually the matrix is chosen to provide the lowest reconstruction error under a specified set of constraints. It results in a constrained matrix factorization problem:

$$\begin{aligned} &\min_{\boldsymbol{M},\boldsymbol{V}} ||\boldsymbol{X} - \boldsymbol{M}\boldsymbol{V}|| \\ &\text{subject to: } \boldsymbol{V}_{:i} \in \mathcal{C} \text{ for all } i = 1, \ldots, N \\ &\qquad \boldsymbol{M}_{:j} \in \mathcal{D} \text{ for all } j = 1, \ldots, p, \end{aligned} \tag{29}$$

where p is the number of features to be found and $\mathcal{D}$ represents the constraints put on individual features, for instance normalization, sparsity, or non-negativity. The optimization problem (29) is often difficult to be solved jointly over $\boldsymbol{M}$ and $\boldsymbol{V}$. Many methods employ instead a form of coordinate descent which alternates between the minimization over $\boldsymbol{M}$ and over $\boldsymbol{V}$.

For example, the popular K-means method represents every data point by a cluster center [35]. Suppose the coordinates of centers form a matrix $\boldsymbol{M}$ in which every column contains the coordinates of a centroid. Assume that the sample $\boldsymbol{x}_i$ belongs to the cluster j. Let v_i be a vector whose only non-zero element is a 1 in the j-th position. Then:

$$\boldsymbol{x}_i \approx \boldsymbol{M}_{:j} = \boldsymbol{M}\boldsymbol{v}_i. \tag{30}$$

If Euclidean distances are used to measure the reconstruction error, the encoding step of K-means is:

$$\begin{aligned} \boldsymbol{v}_i = &\arg\min_{v} ||\boldsymbol{x}_i - \boldsymbol{M}\boldsymbol{v}||_2 \\ &\text{subject to: } \quad \boldsymbol{v} \text{ has only one nonzero element with value 1} \end{aligned} \tag{31}$$

Clearly, the encoding of a given vector $\boldsymbol{x}_i$ is performed by finding the column of $\boldsymbol{M}$ that is closest to $\boldsymbol{x}_i$, that is finding the closest center.

The cluster centers are usually found using the Lloyd's algorithm, which alternates between choosing cluster centers and assigning samples to clusters. It is thus a coordinate descent approach to solving problem (29).

Another approach of this form, the Non-negative Matrix Factorization (NMF) finds an approximate factorization of the data matrix $\boldsymbol{X}$ into two lower-rank matrices that have non-negative elements [56, 57]:

$$\begin{aligned} &\min_{\boldsymbol{M},\boldsymbol{V}} ||\boldsymbol{X} - \boldsymbol{M}\boldsymbol{V}||_F \\ \text{subject to: } &\boldsymbol{M} \succeq 0 \\ &\boldsymbol{V} \succeq 0 \end{aligned} \tag{32}$$

where $||\cdot||_F$ denotes the Frobenius matrix norm and $\succeq$ denotes an element-by-element comparison. Note that the problem defining NMF matches the general formulation (29). The key premise of NMF is that non-negativity of elements prevents complex cancellations between columns of $\boldsymbol{M}$ during reconstruction. Without cancellations only a few terms can enter the sum. Thus sparsity of $\boldsymbol{M}$ is enhanced which results in better understandability of the detected features.

The Sparse Coding problem has similar goals to NMF [58]. It is based on the assumption that the data vectors $\boldsymbol{x}_i$ can be reconstructed using just a few base vectors from a possibly large set which is sometimes referred to as the dictionary. The decoder performs just a matrix multiplication (27). The sparsity assumption is captured by the constraints $\mathcal{C}$. Ideally, feature vectors should be constrained to have few nonzero entries. However, this leads to a hard combinatorial problem. In practice the ℓ_1 norm of the feature vectors in constrained: $\mathcal{C} = \{\boldsymbol{v} : ||\boldsymbol{v}||_1 < \tau\}$. The features are found by solving the problem:

$$\begin{aligned} &\min_{\boldsymbol{M},\boldsymbol{V}} ||\boldsymbol{X} - \boldsymbol{M}\boldsymbol{V}||_2 \\ \text{subject to: } &||\boldsymbol{V}_{:i}||_1 \leq \tau \text{ for all } i = 1, \ldots, N \\ &||\boldsymbol{M}_{:j}||_2 = 1 \text{ for all } j = 1, \ldots, p. \end{aligned} \tag{33}$$

The norm constraint on the columns of $\boldsymbol{M}$ is introduced to prevent degenerate solutions. If large elements of $\boldsymbol{M}$ are allowed, then the elements of $\boldsymbol{V}$ may become arbitrarily small. This in turn makes the sparsity constraint ineffective [58].

The assumption and use of decompositional methods that result in sparse descriptions of signal, images, or patterns has led to many important results in vision research, signal processing, and machine learning. Other works have shown it to be also important in extracting biologically plausible representations of natural images [58, 59]. Furthermore, sparse coding has led to unsupervised induction of parts-based decompositions of data [60, 61].

E Rule Extraction from Black-box Models

Decision trees they are often reformulated into production rules to enhance their understandability [13]. Similarly, one can try to extract the knowledge stored in a black-box classifier. In a more general setting, another learning process is required in which the black-box model serves as an additional information source. This activity is referenced to as rule extraction and formally defined as [62]:

> *Given an opaque predictive model and the data on which it was trained, produce a description of the predictive model's hypothesis that is understandable yet closely approximates the predictive model's behavior.*

Rule extraction has been extensively researched in the domain of neural networks [63]. A useful taxonomy of rule extraction methods is based on how many assumptions are made about the black-box classifier. One class of algorithms (called *pedagogical* in [23] and *independent* in [62]) uses the given classifier solely to make predictions on available and unseen data. This rule extraction scheme makes a direct connection to the PAC learning model defined in the previous section. The black-box classifier is used as the *oracle* function and a probability density estimate of the training data is formed to generate new samples. In the simplest rule extraction scheme new samples are generated and classified using the given black-box model. The additional samples are then used to extend the training set on which a decision tree or production rule learner is induced. An analysis of kernel density estimators, the mathematical state-of-the-art methods for density estimation, is presented in Chapter III.

1 Black-box Independent Rule Extraction

The independent methods do not assume any particular architecture of the black-box classifier. They usually operate by generating artificial samples on which the black-box is queried. They differ in the way the additional samples are generated. Uniform sampling over the range spanned by attributes is conceptually the easiest model of data density. Some authors propose to use it uniquely [64], while others use it as a baseline in comparisons with more advanced methods [65, 66]. Uniform sampling disregards dependencies between attributes. It is often used when samples must satisfy some constraints, as in the ITER [67] and Minerva methods [68].

It is often assumed that the attributes are independent and only the marginal probability distributions are estimated. Just like uniform sampling this approach disregards dependencies between attributes. It is also commonly used to sample under constraints, e.g. TREPAN [66] and DecText [69] both use a Kernel Density Estimator to model the marginal densities. Moreover the TREPAN method sets the kernel width to $1/\sqrt{N}$ where N is the number of samples used for estimation and may build local models for samples that fall into a node of a decision tree.

Many methods that work with discrete data explore Hamming distance balls centered over training samples. The OSRE method treats as important the attributes for which the black-box output changes when the attribute value is changed [70]. The LORE method presented in Chapter IV can be configured to consider samples lying in a Hamming ball around training points. Another rule extraction method based on genetic algorithms first selects a training point, then mutates it [71]. Similarly, the ALBA method first picks randomly and uniformly a support vector. It then adds to every dimension of this support vector a value sampled from a uniform distribution that depends on the mean distance between samples [65].

Rejection sampling is used by the ANN-DT method [72]. First a point is sampled uniformly from the attribute space. It is accepted if it is close enough to a sample in the training set. Hence ANN-DT samples uniformly from the space occupied by the training samples. This is in contrast to multivariate kernel density estimates which assign more probability to regions that are densely occupied by training samples.

Often sampling is intertwined with the white-box learning through the introduction of constraints on generated samples. Many white-box learners operate by dividing the training set into smaller subsets, until a desired purity level is achieved. A common problem with those approaches is that not enough data is available to choose good splitting points/rules as the algorithm progresses. It is thus sensible to generate new samples when such a decision has to be made. The TREPAN method builds a decision tree and generates new samples to decide whether to split a leaf and to possibly choose a split [66]. It uses constraints to only generate samples belonging to that leaf and may use a local density estimate of data at that leaf. DecTex is a similar method which adds new split selection criteria that use the black-box classifier [69]. Similarly the Iter [67] and Minerva [68] method extract non-overlapping rules and generate samples in vicinity of those covered by a rule to decide whether it is possible to generalize it.

2 Black-box Dependent Rule Extraction

In contrast to independent methods, the *decompositional* [23] or *dependent* [62] approaches make direct use of the inner structure of a given black-box classifier. Often this knowledge is used to generate new samples on which the white-box classifier is trained. Craven [73] proposes a sample generator which produces only samples from the positive class by randomly starting hill-climbing optimizers. Khrishnan [74] uses genetic algorithms to generate prototypes, i.e. samples that are assigned by the black-box to a given class with high confidence. He then estimates the distribution of all data samples using a kernel density estimator or a Probabilistic Neural Network [75] and retains only the prototypes that are highly probable given this estimate. On the other hand, many rule extraction methods generate new points close to the decision boundary of the black-box classifier. The Hypinv method finds points on the decision boundary of a Neural Network by a gradient search, however the authors suggest the use of genetic search for non-differentiable models [76]. Similarly, support vectors have been used as the description of an SVM's boundary. Barakat [77] proposes to train an understandable classifier using SVs only, while the aforementioned ALBA method samples points near the SVs [65]. In the case when the black-box classifier is an ensemble of decision trees Domingos proposes to sample from the distribution given by the leaves of trees that form the ensemble [78].

A few techniques were designed to extract from a black-box classifier quantities that are relevant to a newly created white-box one. The structure of a trained neural network is used to select relevant attributes on which a decision tree is grown in [79]. Similarly, a neural network [80] and an ensemble of decision trees [81] are used to estimate information gain needed during the induction of a new decision tree.

Some algorithms for rule extraction, called *purely decompositional*, analyze only the given black-box classifier and disregard the training set. For example, in the case of rule extractions from neural networks those algorithms analyze the network only by finding subsets of inputs causing each neuron to become active [82]. Most of these approaches try to attach an interpretation to hidden neurons of the network, which usually doesn't produce legible results. An exception is the Knowledge Based Artificial Neural Networks [83] in which the network is initialized with prior knowledge forcing an interpretation of all its neurons. This interpretation changes slightly during training and hidden neurons retain their original, meaningful interpretations.

3 Validating Rule Extraction

Criteria commonly used to assess the usefulness of rule extraction methods are: *accuracy* – it measures the ability of the rules to properly classify previously unseen data (generalization ability), *fidelity* – it reflects how well the rules mimic the network, *consistency* – it describes how the rules differ between different training sessions, *comprehensibility* – it states how easy to understand a set of rules is by measuring the number of rules and their antecedents, and finally *computational complexity* – which reflects the needs of the process of rule generation.

So far no rule extraction method has gained wide use and acceptance. In fact, very few implementations of such methods are available (e.g. the large Weka [15] suite provides no such method). Likewise, to the best of our knowledge, no major textbook on Data Mining, Machine Learning, or Neural Networks treats the topic of rule extraction. This may be due to some inherent limitations of the rule extraction process presented in the following section.

F Limitations of Rule Extraction

Rule extraction, or more generally the process of improving the accuracy of an understandable model by introducing an auxiliary black-box model during training usually requires that the black-box classifier consistently outperforms the white-box on the problem at hand. In other words, the model produced by the black-box method must be good enough to warrant the added cost of training it. While this requirement is fairly intuitive, it is often overlooked [65].

Furthermore, the understandability of the white-box classifier must not be sacrificed. Highly precise decision trees or production rules may be too complicated to be understandable. It is often necessary to find a compromise between the complexity of the white-box classifier and its fidelity to the black-box one [84].

Lastly, the rule extraction process must be tractable computationally. For example an algorithm that creates a large truth-table of a given black-box classifier by enumerating all points belonging to the domain is usually too inefficient to be usable [85]. Independent rule extraction algorithms that use the black-box only to provide classifications of new samples must therefore employ sampling of the problem space. It may seem that having access to the internals of the black-box classifier may guarantee lower running times. However, most classifier models are expressible enough to make the rule extraction problem at least as hard as known NP-complete problems. A proof that reduces a satisfiability problem to rule extraction is shown in the forthcoming section.

1 Computation Time Considerations[1]

Before reasoning about the complexity of deriving rules from a trained neural network it must be formally defined what it means to be understandable. The following definition is built on the intuition that when people try to understand a new concept they look for examples of this concept and concentrate on their key properties.

Definition 1. *A rule set is* usable *if it is possible to classify in polynomial time a previously unknown sample. Moreover, it is* understandable *if, for a given class, a sample belonging to that class can be show in polynomial time. If, in polynomial time, the smallest set of features a sample must have to belong to a class can be determined, it will be said that the rule set is* very understandable.

The *usable* rule sets are all those that can be practically used to classify new samples. This is the minimum requirement extracted rules (or any other classifier) must meet to be of any practical use.

The *understandable* rule sets are those, for which examples belonging to a particular class can easily be shown. Decision trees surely meet this criterion – one just has to trace a path from an interesting leaf node to the top of the tree. However, the Disjunctive Normal Form (DNF) formulas are not understandable, because showing examples for the 0 (false) class is equivalent to showing the satisfiability of a Conjunctive Normal Form (CNF) formula (as the negation of a formula expressed in DNF is a formula expressed in CNF), which is a NP-hard problem.

One can argue that the ability to extract examples from the rules is not important, because the data set contains many of them. The *very understandable* class of rules tries to capture the ability of showing the simplest example for a class (or equivalently the shortest clause of the DNF form of the rules). This definition was chosen to resemble some of the actions a person trying to understand an unknown

[1]This section is partially based on the appendix of [86], (J. Chorowski and J. M. Zurada, "Extracting Rules from Neural Networks as Decision Diagrams," *Neural Networks, IEEE Transactions on*, vol. 22, no. 12, pp. 2435–46, Dec. 2011, © 2011 IEEE).

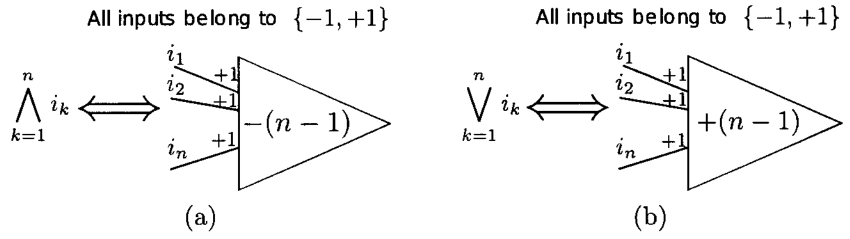

All inputs belong to $\{-1,+1\}$
$\bigwedge_{k=1}^{n} i_k$
i_1
i_2
i_n
$+1$
$-(n-1)$
(a)
All inputs belong to $\{-1,+1\}$
$\bigvee_{k=1}^{n} i_k$
$+(n-1)$
(b)

Figure 3: Implementing the boolean functions (a) *and* and (b) *or* with a neuron. © 2011 IEEE

object would do, i.e. see how it reacts to a given input, find other inputs causing the same reaction and finally, try to find the simplest common factor such inputs may have.

The next two theorems show that in the case of rule extraction from neural networks a purely decompositional approach, which restricts the analysis to the network, leads to a computationally forbidding problem.

Theorem 1. *Extraction of an understandable rule set exactly describing a given neural network is NP-hard.*

Proof. The NP-complete satisfiability of CNF formulas problem will be reduced to the problem of rule extraction from a neural network. The variables of the formula become the inputs. Every hidden layer's neuron implements the logical *or* function and represents a single clause. The output neuron implements the logical *and* function to provide the conjunction of clauses. In Figure 3 it is shown how to express both functions as a neuron.

The described reduction has a polynomial time complexity. If the extracted rules can be used to find in polynomial time an example for which the class is "true", then the rule extraction has to be harder than the satisfiability problem. $\square$

Theorem 2. *It is NP-hard to find a very understandable rule set exactly describing a given network, even if the network's function satisfiability is assumed.*

Proof. A reduction to the dominating-set problem will be demonstrated. As an example, in Figure 4 a neural network corresponding to a small graph is shown. For

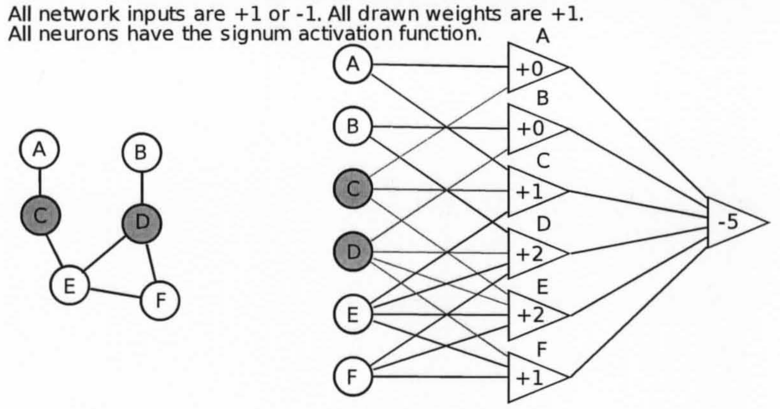

All network inputs are +1 or -1. All drawn weights are +1.
All neurons have the signum activation function.
A
B
C
D
E
F
+0
+0
+1
+2
+2
+1
-5

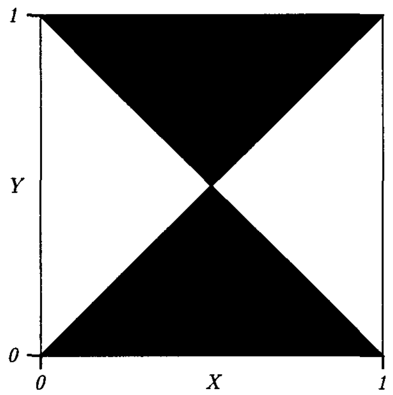
Figure 4: Solving the NP-hard Dominating Set problem can be reduced to finding the smallest set of inputs causing the output neuron to fire. © 2011 IEEE

every node of the given graph, a boolean input feature and a hidden layer neuron is assigned. An input is connected to a hidden layer neuron if and only if there is an edge between the vertexes represented by this input and the hidden layer neuron or if they represent the same vertex. Thus every hidden layer neuron implements an *or* function. The output neuron is then set to implement an *and* function.

Clearly, the function is satisfiable, as an input of all ones (meaning that all vertexes are selected as the dominating set) causes the output neuron to fire. However, finding the smallest set of inputs causing the output neuron to fire is equivalent to selecting the smallest dominating set of graph nodes. □

These proofs may seem to be too artificial to be practically applicable. Usually one can enumerate many points that belong to a particular class just by looking at the training dataset. Furthermore, many classifiers can rank inputs by their significance e.g. [87, 88]. The important observation is however, that the analysis needs to be concentrated on the data itself and points that lie "close" to it.

Another aspect of rule extraction is how easy it is to operate on the data structure used to store the rules. Many algorithms transform and operate on the set of rules, for example the C4.5 Rule has a global rule optimization stage [13]. The expressiveness of the rule language, which is the topic of the next section influences the complexity of algorithmic manipulation of rules. It may be beneficial to select a rule format on which it is easy to operate. Reduced Ordered Decision Diagrams,


1
Y
0
0
X
1
if X-Y>0 and X+Y<1 then white
if X-Y<0 and X+Y>1 then white
else black

(a)

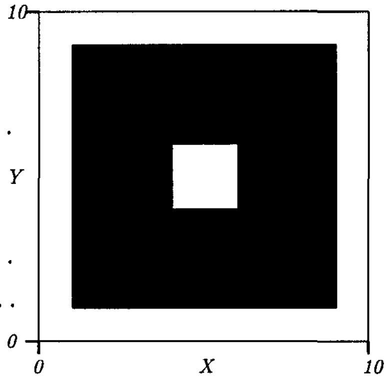
Using M-of-N approach:

if more_than 3 *out_of*
(a, b, c, d, e, f) *then*...

Without M-of-N:

if a and b and c then...
if a and b and d then...
$\vdots$
if d and e and f then...

(b)


10
Y
0
0
X
10
Using exceptions:
if $X \in (1,9)$ *and* $Y \in (1,9)$ *then* black
unless $X \in (4,6)$ *and* $Y \in (4,6)$
else white

(c)

Figure 5: Expressive power of different types of expressions used in rules: (a) oblique rules involving linear inequalities (after [89]), (b) M-of-N tests, and (c) rules with exceptions (a description without exceptions would require at least 4 rules for the class `black`).

presented in Chapter IV, Section A, are a data structure that allows to succinctly express many concepts while allowing their efficient algorithmic manipulation.

2 Concept Representation Considerations

Many black-box models are able to delineate concepts by highly irregular and complicated decision boundaries. Multilayer neural networks operate by dividing the space using hyperplanes smoothed out with sigmoidal functions. On the other hand, Support Vector Machines use kernel functions to project the samples into a highly dimensional space in which they look for a separating hyperplane. Likewise, ensembles can express concepts that are impossible to describe with their base learners.

Rule extraction methods should, on the one hand, have similar abilities to concisely express complex concepts. On the other hand, the use of highly expressible constructs can hinder their understandability. For instance, linear inequalities, also called oblique rules (Figure 5a) have been found very difficult to understand [89] but at the same time they are not easily expressed using other concepts.

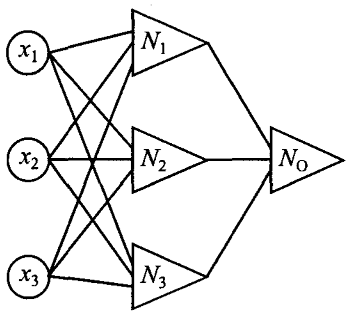
Popular approaches that extend the rule language to increase its succinctness and comprehensiveness, extend the allowed language to support M-of-N tests (Figure 5b) and exceptions (Figure 5c). The former adds the ability to express conditions which are true if more than M, less than M, or exactly M conditions are true out of N given. This is especially effective in the case of rule extraction from neural networks [83]. The later allows to enrich rules with exceptions to them, instead of trying to create non-overlapping rules. Attributional Calculus is a rule language that extends propositional calculus with attribute types, hierarchical attributes, exceptions, and M-of-N selectors [90].

There are two dangers related to the increase of the expressibility of the rule language. Firstly, increasing the expressibility of the rule language effectively increases its Vapnik-Chervonenkis dimension. Thus more training data is needed to select a good rule set. Secondly, even small sets of rules expressed in a powerful language may be hard to understand. For instance during rule extraction from a neural network one may introduce artificial variables that correspond to the state of hidden neurons. It is then easy to write down DNF rules for the state of hidden neurons in terms of the inputs and for the output of the network in terms of the hidden neurons. However, such a rule set may be difficult to comprehend when the hidden neurons do not represent meaningful concepts.

To illustrate this problem a neural network with three hidden neurons was trained to recognize 3-bit parity. The hyperbolic tangent transfer function was used. Furthermore, the inputs and outputs were scaled to ± 1. Network architecture and functions computed by the neurons after training are depicted in Figure 6a.

The logical rules describing the neurons are shown in Figure 6b. They assume that the neurons' outputs are always ± 1, which can be accomplished by replacing the hyperbolic tangent activation function with a hard threshold one. Investigation of individual neurons doesn't provide a good understanding of the network's operation. Moreover, despite symmetry of the problem, the input x_2 is singled out and appears to be more important.

The network is described by four rules containing in total 10 clauses. Even



Neuron	Weights
N_1	$0.0x_1 + 3.2x_2 + 0.0x_3 - 1.9$
N_2	$3.1x_1 - 2.3x_2 + 3.1x_3 - 0.3$
N_3	$-2.8x_1 - 2.2x_2 - 2.9x_3 + 0.1$
N_O	$4.3\sigma(N_1) + 4.3\sigma(N_2) + 4.3\sigma(N_3) + 0.3$

(a) Network architecture and functions implemented by neurons.

Neuron	Function (DNF)	Function (M-of-N)
N_1	x_2	1 of $\{x_2\}$
N_2	$(x_1 \wedge \neg x_2) \vee (x_1 \wedge x_3) \vee (\neg x_2 \wedge x_3)$	at least 2 of $\{x_1, \neg x_2, x_3\}$
N_3	$(x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$	at least 2 of $\{x_1, x_2, x_3\}$
N_O	$(N_1 \wedge N_2) \vee (N_1 \wedge N_3) \vee (N_2 \wedge N_3)$	at least 2 of N_1, N_2, N_3

(b) Logical description of individual neurons.

Figure 6: Incomprehensibility of a small Neural Network: (a) architecture of the network and functions implemented by neurons; (b) logical description of neurons under a threshold activation assumption.

though the rule set is small, it is not fully understandable. The symbols introduced for the hidden neurons are not related to the concept of parity. The function of the network becomes meaningful only after the output is expressed directly in terms of the inputs:

$$N_O = (x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge x_3)$$

However, the substitution of the representations of the hidden neurons into the formula describing the output neuron may require a consideration of all combinations of hidden neuron states, which often is intractable computationally. Furthermore, when a smooth activation function is used the hidden neurons must not be treated as binary variables, further complicating the task. This shows that while describing a network by concentrating on single neurons may lead to a concise formulation, the understanding of the network requires a holistic approach that may be prohibitively time consuming.

This example of opacity of rules extracted from neural networks relies on the expressiveness of hierarchical models coupled with their opaqueness when the concepts used in the hierarchy are not easily interpretable. However, if interpretable features could be derived from data, then they should be used for the creation of understandable hierarchical models. An attempt at enhancing the understandability of hidden neurons through the introduction of non-negative weight constraints has been presented in Chapter V, Section A.

3 Do Black-box Classifiers Learn Knowledge from Data?

According to the Statistical Learning Theory, a learning machine is trained to approximate the outputs of the supervisor. The main goal is to imitate, not identify, the supervisor's behavior. Does it still imply that the learning machine gains knowledge about the supervisor's operation? Knowledge is not formally defined and just like the notion of understandability it is very subjective. For example, the nearest-neighbor classifiers assumes that the supervisor's output varies little between similar inputs. The predicted class is just the most popular class of a specified number of neighbors of a testing point. Does a nearest neighbor classifier learn new knowledge? Intuitively no, because any analysis of a nearest neighbor classifier is essentially an analysis of the raw training set.

Neural Networks may seem closer to extracting useful relations out of the training data. A multilayer feedforward neural network implements a complicated function of the inputs and weights. It is trained by modifying the weights. Unlike a nearest neighbor classifier, the size of a neural network is usually fixed and doesn't grow with an increase of the number of processed training samples. However, as demonstrated by the example in the previous section, those characteristics do not guarantee that the weights of a network are meaningful. In fact, an artificial neural network may work consistently well even when most of its weights are chosen at random. This is proposed by the Extreme Learning approach [91].

An Extreme Learning Machine (ELM) is a feedforward neural network with one hidden and one output layer. However, only the weights and biases in the output layer are optimized, while the weight and biases of the hidden layer are generated randomly and kept fixed during training [91]. Thus they bear no relation to the problem and can not capture any knowledge. Neither they possess any meaningful interpretation. However, it has been verified that the accuracy of an ELM is often comparable to the state-of-the art classifiers, such as Support Vector Machines [92].

The examples of nearest neighbor classifiers, or of neural networks show that unless care is taken, even a very accurate classifier may be oblivious to much of the data structure. The task of learning an understandable classifier may then necessitate conscious engineering of the classifier to make it "knowledgeful." This can be accomplished by using known data properties or invariants explicit in the classifier design.

The Knowledge-Based Artificial Neural Networks (KBANN) methodology was proposed to fulfill this goal [93]. It describes how to encode rules into a neural network, tune the network, and finally read the refined rules from the network. At the beginning each neuron encodes a single rule. Network tuning changes the neurons' weights slightly, causing the related rules to be refined, but not redefined. This means that after tuning individual neurons can be analyzed separately and meaningful rules can be extracted from the network.

In many cases the knowledge of a problem is not limited to input-output pairs and additional properties are known. Often it is known that the classifier should be insensitive to some transformations of the data, that will be henceforth referred to as "invariants". For instance in a system designed for handwriting recognition small translations or rotations of characters do not change the text being recognized. Other kinds of information may be known too. The hypothetical handwriting recognizer might also know the language of the text. How can this additional information be incorporated into the data mining process?

One way of using the information about a particular problem is a judicious selection of the training data. Proper description of a problem through a selection of relevant attributes is crucial. Often the learning itself is not applied to the raw data, but to a cleaned and hand-curated subset of it. In many domains specialized data transformations are routinely used. For instance audio processing usually begins with the computation of the cepstrum of the signal [94]. For computer vision tasks the

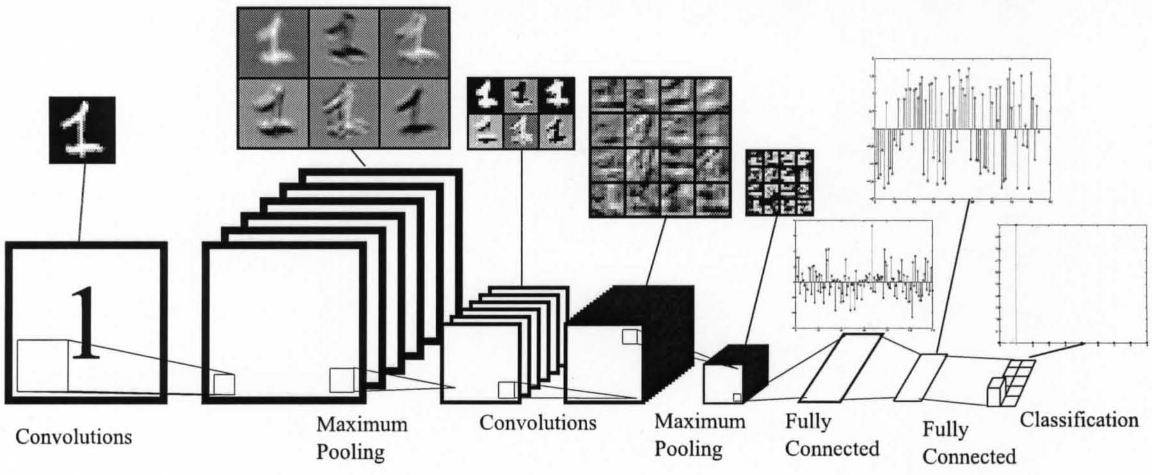


Figure 7: Signal propagation through a convolutional network to recognize handwritten digits. The network architecture is based on LeNet [97]. The images of network's operation are obtained using a multidimensional convolutional network implemented for comparison of methods for 3D action recognition [98].

SIFT features [95] or local binary patterns [96] provide descriptors that are invariant under many image transformations. Furthermore, the data set can be extended with artificial samples that are special transformations of the original set. This technique is often used in the domain of text recognition, where the training set is extended with geometrical distortions of existing samples [97].

Once the training data are selected, the learning machine itself can be designed to reflect some of the known data invariants. Arguably, the largest possibilities are offered by neural network based approaches, because at its core a neural network is a nonlinear function with many tunable parameters. Desired behavior can then be obtained through a reformulation of this function and through the addition of regularization terms that promote a particular behavior of the classifier.

A case in point, Convolutional Neural Networks (CNNs) excel at character recognition [97]. They are based on the idea that weights used to recognize small features, from which individual characters are composed, are probably the same at any location of a larger input image. To exploit this observation, a CNN uses neurons that are filters to be convolved with the input. Such neurons do not produce a single activation value, but activation maps. The size of those maps is reduced with pooling operators, that aggregate neighboring outputs into a single value. An architecture of a CNN for handwritten digit recognition is presented in Figure 7. It shows schematically the convolution and pooling operations, as well as snapshots of the signal propagated through the network. The CNN begins with a sequence of convolutional and pooling layers, which is followed by a classical multilayer perceptron network. In this way properties of text images are incorporated in the classifier architecture. Convolutions capture the property that detectors for basic features in an image do not vary with the location, while insensitivity to small translations of the input is achieved through pooling.

The architecture of Convolutional Neural Networks was inspired by the discovery of the *simple* and *complex* cells in the visual cortex of a cat [99]. In particular the filters used for convolution correspond to simple cells that detect single patterns. On the other hand, the pooling operation aggregates filter responses over neighboring locations and is related to the complex cells. Other studies were directed at explaining which mechanisms lead the formation of receptive fields of simple cells in the visual cortex. It has been shown that optimizing sparsity of image representation produces oriented Gabor-like filters that agree with those found in the macaque visual cortex [58]. In turn the use of signal sparsity in a neural network based classifier led to state-of-the art accuracies [60] on handwritten digit recognition.

The introduction of sparsity has important consequences for the understandability of the learned model. A signal representation is sparse when the signal is represented by only a few elements from a large set. Especially on vision tasks this often leads to a hierarchical decomposition of the input into meaningful parts. A decomposition of digits into pen strokes was presented in [100] and a decomposition of common objects into parts was demonstrated in [101]. Finding sparse encodings of signals in terms of a known dictionary is also an important topic. An efficient algorithm that finds robust and sparse encodings of data is described in Chapter V, Section B.

Knowledge of other invariants can be included in the learning machine's design by forcing parts of its output or state to be constant under some data

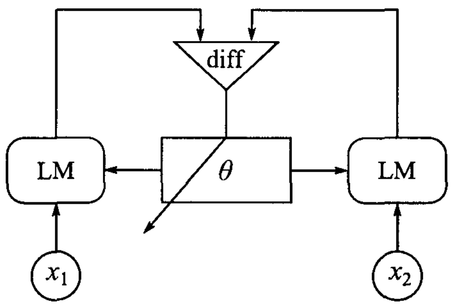


Figure 8: Making the Learning Machine LM aware of an invariant about data. x_1 and x_2 are two inputs for which LM's output should be the same. The output of the LM is computed twice using the same parameters Θ. The difference between the two outputs is computed and Θ is adjusted to minimize it.

transformations. Usually two steps are required, as depicted in Figure 8. First, pairs of inputs for which the output should be similar are generated. The learner is trained to minimize the difference between outputs produced from those pairs [102]. In this way the learner is forced to learn an invariance relation. This approach can also be used to learn a transformation that reduces the dimensionality of data. The learner's output maps the data into a space that preserves known invariants [103].

Sometimes constraining the representation used by the learner is enough to generate a meaningful representation. The Non-negative Matrix Factorization (NMF) algorithm postulates to represent an input as a non-negative combination of non-negative basis vectors [57]. This restriction suppresses cancellation between base vectors, thereby enhancing sparsity. In contexts in which non-negativity is meaningful, such as in analyzing counts of words in documents or pixel intensities in images a simple parts based representation can be found. Results of an application of the non-negativity principle to feed-forward neural networks are presented in Chapter V, Section A.

Examples presented in this section demonstrate how a classifier can be designed to match known data characteristics. Those techniques may make the classifier more understandable, because its design will be in better agreement with knowledge about the problem. However, this brings the task of learning understandable models further away from a universal approach that does not require extensive modeling and closer to the parametric statistical approach. Perhaps a fully automatic method does not exist and the user must guide the learning process to obtain a model that will enhance his subjective understanding of the problem at hand.

G Two Facets of Learning Rules

Learning machines are trained to approximate an unknown function from a set of inputs into a set of discrete class labels. Sometimes, this relation can be expressed in a "shallow" way in which the outputs are simple combinations of the inputs. In fact, the majority of rule learners build a list of rules which are conjunctions of simple tests of inputs. The shallow models have obvious limitations on the complexity of concepts they can represent. On the other hand, they are often easy to interpret.

The hierarchical, or "deep" models employ a sequence (hierarchy) of transformations of the data. Complex concepts are built out of simpler ones. However, when the intermediate concepts are not understandable, the whole hierarchy becomes opaque. Automatic induction of hierarchical models, that do not guarantee understandability, is easily achievable. One can for instance train a multilayer feedforward neural network and treat each layer as a level of the hierarchy. However, as the example given above demonstrates, understandability of such a model is usually very low. Moreover, since there is no definition of understandability it is not possible to directly optimize a model to be more understandable. A possible solution of this problem consists of building the hierarchy one level at a time. This introduces the problem of feature detection. It consists of finding a data transformation that transform raw input samples into a new form that contains useful and understandable features.

The two facets of rule extraction are linked to those two different regimes of operation. On the one hand it is important to construct accurate shallow models. However, those models may refer to features extracted from the data, and not to the original attributes used for data description. This dissertation treats both topics. Chapters III and IV describe methods for efficient extraction of shallow models from data and black-box classifiers. Chapter V describes how understandable and useful for classification features may be found. It also provides an efficient algorithm for finding sparse descriptions of the data.

CHAPTER III

IMPROVING THE ACCURACY OF WHITE-BOX MODELS THROUGH ADDITIONAL SAMPLE GENERATION[1]

This chapter evaluates a family of very intuitive rule extraction methods in which a white-box classifier is trained on artificial data sampled from an estimate of the probability density of the original data. As such the described method can be used with any black-box classifier and belongs to the family of independent or pedagogical rule extractors.

The independent methods were originally introduced under the name of *rule extraction as learning* [73]. The generic algorithm is pictured in Algorithm 1. The important question is how to choose a proper method of generating additional samples in line 2 of the algorithm that would yield maximum increase in the white-box learner performance, without sacrificing too much of its understandability. A review of pedagogical rule extraction methods that generate additional samples was presented in Chapter II, Section E. Many of those methods use very simple, or ad-hoc methods of density estimation. This chapter describes results obtained when state-of-the-art mathematical techniques for density estimation are used instead.

Artificial sample generation is often used to increase the accuracy of classifiers. Text recognizers are often trained on a data extended with geometric transformations of samples [97]. Similarly, the SMOTE method mutates existing samples to aid learning in the presence of class imbalance [105]. The rule extraction

[1]This chapter is based on [104] (J. Chorowski and J. M. Zurada, "Improving the accuracy of understandable classifiers through additional sample generation," *Submitted to Knowledge and Data Engineering, IEEE Transactions on*, 2012).

Algorithm 1: General schema of rule extraction as learning methods.

```
1: $BB \leftarrow$ a black-box classifier trained on $TrainData$
2: $NewData \leftarrow$ Additional samples
3: Classify $NewData$ using $BB$
4: $WB \leftarrow$ a white-box classifier trained on $TrainData \cup NewData$
```

as learning approach differs by the assumption that a reliable black-box classifier is available to label new data. Therefore, since the labels will be set using the black-box, data transformations used to generate new samples can generate samples belonging to other classes than their seed samples.

Chapter II argues that classifier accuracy measured on unseen (test) data increases with an increase of the number of training samples. This motivates the rule extraction as learning approach whose main idea is to extend the training set with additional artificial samples.

Another intuitive motivation comes from analysis of the operation of understandable classifiers. Decision trees are usually built by repeatedly splitting the training set. Similarly rule learners often employ the *separate and conquer* approach which repeatedly finds a rule covering some samples which are then removed (separated) from the training data. Tree splits and rules are often selected by maximizing a statistical measure of their performance. As algorithms progress, the data becomes scarce. Therefore some patterns may be missed by the learner and other patterns may be selected purely by chance. The generation of additional training data should therefore help the algorithms to learn a more complete theory that governs the data at hand.

The remaining question is what distribution should the new data come from. If a global understanding of the black-box classifier is required, then the white-box should ideally replicate black-box's responses in the whole space spanned by data attributes. For small domains it was proposed to generate a full truth table and simplify it using e.g. Karnaugh maps [85]. For large domains this direct approach requires prohibitively many computations and uniform sampling may be necessary. However, trying to replicate the behavior of the black-box classifier over the whole space has several disadvantages.

When the end goal is to obtain better accuracy on unseen testing data, it makes little sense to model the behavior of the black-box in regions where data are scarce. Performance measures such as accuracy are defined on a testing set that is assumed to be drawn randomly and independently from the same distribution as the training set. Similarly, rule extraction methods are often characterized by their *fidelity* to the black-box classifier, which is defined as the percentage of the testing samples for which the black-box and the white-box gave the same answer [23]. Moreover, both SLT and PAC learning theories require that the training and testing data come from the same, albeit unknown, probability distribution. This suggests that new data should be drawn from the original data density.

Furthermore, when new data comes from the distribution of the original data, the resulting white-box classifier may be simpler because it is not forced to approximate the decision surface of the black-box classifier in areas not populated by the data [66]. Moreover, fewer additional data samples may be needed to attain a certain level of accuracy. It becomes clear that it is beneficial to generate additional samples from the distribution of the original data itself. The remaining question is how to do it.

Suppose that the training set contains N pairs $(\boldsymbol{x}_n, y_n)$ formed by a vector of attribute values $\boldsymbol{x} \in \mathbb{R}^k$ and a class label y. Assume that the training samples are drawn i.i.d. from a probability distribution P_{XY}:

$$X, Y \sim P_{XY}. \tag{34}$$

The probability of observing a data sample $(\boldsymbol{x}, y)$ can be expressed as:

$$P(Y = y, X = \boldsymbol{x}) = P(Y = y|X = \boldsymbol{x})P(X = \boldsymbol{x}). \tag{35}$$

This factorization directly corresponds to the data generator and supervisor assumed by the statistical learning theory and show in Figure 1 in Chapter II. Classifier training consists of imitating the supervisor and approximating the conditional probability $P(Y = y|X = \boldsymbol{x})$. Rule extraction requires that the black-box classifier has learned a good approximation of it. Experiments reported at the end of this chapter demonstrate that good accuracy of the black-box is required to achieve satisfactory results. The remaining issue is to estimate the distribution over the attributes $P(X)$. This is the main topic of this chapter.

The Algorithm (1) does not apply the formula (35) exactly because labels are deterministically set based on the black-box predictions, rather than sampled from the predicted class distribution. This is done for two reasons. First, it biases the white-box toward learning the decision boundary of the black-box. Second, many practically used classifiers do not compute class probabilities, only the decision. For instance the SVM must be extended in nonstandard ways to provide approximate probabilistic interpretations of its outputs [106, 107].

A Review of Probability Density Estimation Methods

The problem of nonparametric density estimation has been well researched [108, 109]. One family of non-parametric density estimators is the *kernel estimators* also known as *Parzen windows*. It uses locally tuned radial basis functions (i.e. kernels) to interpolate the density function between observed samples. Another family of methods first finds several 1D projections of the data using *projection pursuit* or a similar technique. New samples are generated in this transformed feature space. The two families have been compared in [110].

1 Kernel Density Estimation

Given a set of N k-dimensional training data $\{\boldsymbol{x}_n, n = 1, \dots, N\}$, a multivariate *fixed-width kernel density estimator* (FKDE) with the kernel function K and a fixed (global) kernel width parameter h, gives the estimated density $\hat{f}(\boldsymbol{x})$ for a multivariate data $\boldsymbol{x} \in \mathbb{R}^k$ based on [110]:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{Nh^k} \sum_{n=1}^{N} K\left(\frac{1}{h}(\boldsymbol{x} - \boldsymbol{x}_n)\right), \tag{36}$$

where the kernel function K satisfies [110]:

$$K(\boldsymbol{x}) \geq 0, \quad \text{and} \int_{\mathbb{R}^k} K(\boldsymbol{x}) d\boldsymbol{x} = 1. \tag{37}$$

A popular choice of K is the Gaussian kernel [110]:

$$K(\boldsymbol{x}) = (2\pi)^{-k/2} \exp\left(-\frac{1}{2}\boldsymbol{x}^T\boldsymbol{x}\right), \tag{38}$$

which is symmetric with its value smoothly decaying away from the kernel center.

The width of the kernel h is an important parameter of FKDE. If it is too small, the estimated density will have spikes at data-points and wrongly estimate the density in the tails of the distribution. For the width too large, the estimate will loose some of the structure of the original distribution.

There are simple methods to choose an initial value of the kernel width. A popular method suitable for univariate data and Gaussian kernels assumes that the original distribution is normal, compensating for the case when the experimental inter-quartile range is too small. The kernel width is given by [108]:

$$\begin{aligned} A &= \min(\text{std. dev.}, \text{inter-quart. range}/1.34) \\ h &= 1.06AN^{-1/5}. \end{aligned} \tag{39}$$

This method can be extended to the multivariate case by first sphering[2] the data using [110]:

$$\boldsymbol{z} = \mathcal{S}^{-1/2}(\boldsymbol{x} - E\{\boldsymbol{x}\}), \tag{40}$$

where $\mathcal{S}$ is the observed data covariance matrix and E denotes the mathematical expectation operator:

$$\mathcal{S} = E\left\{(\boldsymbol{x} - E\{\boldsymbol{x}\})(\boldsymbol{x} - E\{\boldsymbol{x}\})^T\right\} = \boldsymbol{UDU}^T \tag{41}$$

$$\mathcal{S}^{-1/2} = \boldsymbol{UD}^{-1/2}\boldsymbol{U}^T \tag{42}$$

The FKDE of sphered data is given by:

$$\hat{f}(\boldsymbol{z}) = \frac{1}{Nh^k}\sum_{n=1}^{N} K\left(\frac{1}{h}(\boldsymbol{z} - \boldsymbol{z}_n)\right) \tag{43}$$

[2]Sphering of data is also known as Zero Component Analysis.

After sphering $E\{\boldsymbol{z}\} = 0$ and $E\{\boldsymbol{z}\boldsymbol{z}^t\} = \boldsymbol{I}$ (the identity matrix). Under the assumption that the sphered data are normally distributed and that the kernel is multivariate normal the optimum width is [108, 110]:

$$h = AN^{-\frac{1}{k+4}}, \text{ where } A = \left(\frac{4}{2k+1}\right)^{\frac{1}{k+4}}, \tag{44}$$

and k is the number of dimensions.

A product of univariate kernels K can also be used for multivariate FKDE. The FKDE with a *product kernel* is defined as [109]:

$$\hat{f}(\boldsymbol{y}) = \frac{1}{Nh_1 \cdots h_k} \sum_{n=1}^{N} \left(\prod_{j=1}^{k} K\left(\frac{\boldsymbol{y}_j - \boldsymbol{x}_{nj}}{h_j}\right)\right). \tag{45}$$

Furthermore when the kernel K is Gaussian, the widths can be set to [109]:

$$h_j = (\text{std. dev. in dimension } j) \cdot N^{-\frac{1}{k+4}} \tag{46}$$

More sophisticated methods for choosing and tuning the kernel width have been proposed [108, 111, 112]. Often the methods are iterative and optimize the kernel width by estimating the goodness of fit of the distribution using cross-validation. Their detailed description is omitted, because the proposed rule extraction algorithm only uses density estimation as a step toward obtaining a more understandable and accurate classifier. As such, the end-goal is not a perfect density estimate, but good accuracy of the white-box classifier. It may be more advantageous to use (39) as a starting point and fine-tune the width by cross-validating the white-box classifier accuracy.

Extensions of Kernel Density Estimation

The kernel width can be allowed to vary among samples, making it possible to adjust the smoothness of the kernel estimate to the amount of available data. The resulting technique is called *Adaptive Kernel Density Estimation.* It can be constructed in two steps, by first selecting a pilot estimate of the data, then using it to compute the individual kernel widths [108, 110].

If the data set is large fewer kernels may be used than there are data samples. It is also possible to relax the requirement that the kernels are placed on data samples only. In this way kernel density estimation essentially results in a *Radial Basis Function* network, or a *Gaussian-Mixture Model* [36, 110].

Handling Discrete Data

Estimation of univariate discrete data is easily accomplished by computing the observed frequencies of different values. In the case of multivariate k-dimensional discrete data a distance measure between samples has to be introduced. A suitable one is the Hamming distance [108]:

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{k} \mathcal{I}\{\boldsymbol{x}(i) \neq \boldsymbol{y}(i)\}, \tag{47}$$

where $\mathcal{I}\{\cdot\}$ is the indicator function and $\boldsymbol{x}(i)$ denotes the value of the i-th attribute of sample $\boldsymbol{x}$. When all attributes are binary, for any λ with $1/2 \leq \lambda \leq 1$ a kernel K_B is defined as [108]:

$$K_B(\boldsymbol{y}|\boldsymbol{x}, \lambda) = \lambda^{k-d(\boldsymbol{x},\boldsymbol{y})}(1-\lambda)^{d(\boldsymbol{x},\boldsymbol{y})}. \tag{48}$$

The value $k\lambda$ can intuitively be interpreted as the average number of disagreements between samples.

In practical scenarios attributes that take more than two values need to be considered. Furthermore, the original marginal probabilities of attributes should be preserved by the extended data set. Therefore this study proposes the kernel:

$$K(\boldsymbol{y}|\boldsymbol{x}, \lambda) = \prod_{i=1}^{k} \Big[\Big(\lambda + (1-\lambda) P_i\big(\boldsymbol{y}(i)\big)\Big)^{\mathcal{I}\{\boldsymbol{x}(i)=\boldsymbol{y}(i)\}} \cdot \Big((1-\lambda) P_i\big(\boldsymbol{y}(i)\big)\Big)^{\mathcal{I}\{\boldsymbol{x}(i)\neq\boldsymbol{y}(i)\}} \Big], \tag{49}$$

where $P_i(v)$ is the experimental marginal probability of the i-th attribute taking the value v and λ is a parameter in the range $[0, 1]$ that controls the amount of smoothing. This kernel is more complicated than K_B, however it has a simple interpretation and allows for a very intuitive sampling algorithm shown in Algorithm

2. The intuitive meaning is that for every attribute with probability λ the value of this attribute is selected randomly from the respective marginal distribution.

For both kernel choices the approximate density function is defined by [108]:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N} K(\boldsymbol{x}|\boldsymbol{x}_i, \lambda) \tag{50}$$

There is no simple formula to choose an appropriate value for λ, however cross-validation may be used to choose the λ^* that maximizes the score [108]:

$$\lambda^* = \arg\max_{\lambda} \sum_{i=1}^{N} \log \hat{f}_{-i}(\boldsymbol{x}_i), \tag{51}$$

where $\hat{f}_{-i}$ denotes the estimate obtained from all data points with the exception of point $\boldsymbol{x}_i$.

When the data contains both numerical and nominal attributes, an FKDE can be constructed with the kernel K defined to be the product of a kernel for the continuous attributes K_c and a kernel for the nominal ones K_n. This approach requires the specification of two parameters: h for the continuous kernel and λ for the nominal one.

Another possibility of handling discrete attributes is to transform them into numerical ones and use a standard FKDE estimator. The *Weight of Evidence* (WoE) transformation has been proposed for use in rule extraction in [65]. It is often a good choice because for problems with two classes it avoids the creation of artificial variables. In a problem with two classes c_1 and c_2 the WoE transformation of the value v of attribute A is defined as follows:

$$WOE(A = v) = \ln \frac{P(C = c_1|A = v)}{P(C = c_2|A = v)}. \tag{52}$$

The WOE score is positive when there are more samples of class c_1 having value v of attribute A than there are samples of class c_2. In multi-class problems one can introduce WOE scores for every possible class value:

$$WOE_j(A = v) = \ln \frac{P(C = c_j|A = v)}{P(C \neq c_j|A = v)}. \tag{53}$$

Algorithm 2: Drawing a sample from a FKDE.

```
1: Choose $i$ uniformly with replacement from $\{1,\dots,N\}$
2: if data are continuous then
3:   Generate $\epsilon$ to have probability density $K$
4:   return $\boldsymbol{x}_i + h\epsilon$
5: else {Sample $\boldsymbol{x}_n$ conditioned on $\boldsymbol{x}_i$ from probability density given
   by the kernel (49)}
6:   $\boldsymbol{x}_n \leftarrow \boldsymbol{x}_i$
7:   for all $a \in$ attributes do
8:     if $\lambda < U(0,1)$ then
9:       $\boldsymbol{x}_n(a) \leftarrow$ random value of attribute $a$ generated according to
         its marginal distribution
10:    end if
11:  end for
12:  return $\boldsymbol{x}_n$
13: end if
```

When the conditional probabilities are estimated form the data it is possible that no data exists for a particular class and attribute value combination. Direct computation of the WoE score may thus result in a division by zero error. In experiments this condition was prevented through the use of Laplace smoothing (occurrence counts of attribute value and class combinations are increased by 1, thus ensuring that no count is 0).

Generating Samples From a FKDE

In this application the algorithm uses the density estimate to generate new data. When FKDE is used it is not needed to explicitly construct the estimate. Suppose a FKDE estimate $\hat{f}$ is obtained from samples $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$. The classical algorithm designed for continuous attributes [108] is extended to also handle discrete data. Suppose that a kernel K of width h is used for continuous attributes and that the kernel (49) with smoothing parameter λ is used for the discrete ones. An algorithm to sample from $\hat{f}$ is presented in Algorithm 2. The algorithm is easy to implement when Gaussian kernel is used for numeric attributes. Moreover it is not required to explicitly evaluate the density estimate.

2 Projection Pursuit-based Methods

Projection pursuit looks for "interesting" low dimensional data projections. Usually, non-gaussianity of the projected data is optimized to find good projections. After a projection direction is found, the structure of the projection is removed form the data to make it "less interesting" and enable finding other projections. This process can be inverted to form a sampling scheme in which the data is sampled in the reduced space of the projections, then the structure is iteratively added [110].

Independent Component Analysis (ICA) of the data, which also searches for projections that maximize non-gaussianity of the data [55, 113] can also be used to find interesting projection directions. In the ICA space independent univariate kernel estimators can be used to model the density of individual ICA features. New samples are generated from those estimates. Finally, the inverse transformation is performed to back-project the generated samples into the original attribute space.

B Experiments

This section compares the accuracy and size of white-box classifiers induced on original and artificial data generated using various density estimation methods. There is little point in evaluating rule extraction approaches on data sets on which the white-box methods already excel [65]. Therefore the comparison was executed on a selection of six data sets from the UCI repository [17] for which the white-box classifiers were consistently less accurate than the black-box ones. The properties of the selected data sets are given in Table 1. The "Vote" data set has been made more difficult by the removal of the attribute "physician fee freeze". Samples containing missing values were discarded because missing values are not handled by the LibSVM solver [114].

For reference the results of using two popular black-box classifiers to label the generated samples were compared: The "Fast Random Forest" Java implementation [16] of the Random Forest (RF) [7] and the LibSVM [114] implementation of the Support Vector Machine (SVM) [8]. Both classifiers were

TABLE 1

Properties of data sets used in experiments.

Data Set		Number of			
Full Name	Abbr. Name	Inst.	Nom. Attrs.	Num. Attrs.	Classes
Balance Scale	Bal	625	0	4	3
German Credit	Ger	1000	13	7	2
Sonar	Sonar	208	0	60	2
Promoter	Prom	106	57	0	2
Vote	Vote	232	15	0	2
Wine	Wine	178	0	13	3

accessed from Matlab. The RF was configured to always build 100 trees with all other parameters set to their default values. The SVM was configured with a Gaussian kernel. Moreover the parameters γ (SVM kernel width) and C (regularization) were optimized to yield maximum cross-validation accuracy using grid search ($\gamma : 2^{-15}, 2^{-13}, \dots, 2^{3}$, $C : 2^{-5}, 2^{-3}, \dots, 2^{15}$).

For reference purposes, also two white-box learners, both implemented in the Weka data mining suite, were used for comparison: the JRip implementation of the RIPPER production rule learner [14] and the J48 implementation of the C4.5 decision tree learner [13]. JRip was used with its default settings, while J48 was used with pruning performed either using the pessimistic confidence interval heuristic (the default) and using Reduced Error Pruning (REP) in which the tree is grown and pruned on different parts of the training set. Sometimes enabling REP resulted in the induction of much smaller trees.

In all experiments 25 runs of 10-fold Cross-Validation (CV) were performed. That many runs are necessary because the white-box classifiers are unstable – small variations of the training data lead larger accuracy variations. The results obtained on the original data are gathered in Table 2 which shows the mean and standard deviation of CV accuracies obtained by all classifiers. Moreover, for white-box classifiers the accuracies when the original training labels were replaced with black-box predictions are also reported. It can be observed that replacing the labels

TABLE 2: Cross-validation classifier accuracies on original data.
Means and standard deviations (in parenthesis) of 10xCV accuracies from 25 experiment runs. The white-box accuracies are reported on the original data set, and on the training data with labels replaced with either of the black-box's predictions. The "+WoE" suffix in data set name indicates that nominal attributes were processed with the weight-of-evidence filter.

Data	SVM	RF	JRip Acc			J48 Acc			J48+REP Acc		
Name	Acc	Acc	Orig	SVM	RF	Orig	SVM	RF	Orig	SVM	RF
Bal	99.9% (0.1)	81.2% (0.6)	80.8% (0.9)	80.8% (0.9)	80.8% (0.9)	77.9% (0.9)	77.9% (0.9)	77.9% (0.9)	78.2% (1.1)	78.2% (1.1)	78.2% (1.1)
Ger+WoE	76.2% (0.3)	76.5% (0.7)	73.1% (1.0)	73.5% (0.8)	73.2% (0.9)	72.0% (0.9)	74.3% (0.9)	72.0% (0.9)	72.2% (1.0)	73.6% (1.0)	72.2% (0.9)
Ger	75.9% (0.8)	75.9% (0.7)	72.2% (1.0)	72.0% (0.9)	72.2% (1.0)	71.1% (0.7)	73.1% (0.9)	71.1% (0.7)	72.0% (1.0)	73.2% (1.0)	72.0% (1.0)
Prom+WoE	98.2% (0.8)	93.8% (1.7)	82.0% (3.2)	82.1% (3.2)	82.0% (3.2)	77.0% (2.4)	77.1% (2.4)	77.0% (2.4)	76.8% (3.3)	76.7% (3.2)	76.8% (3.3)
Prom	92.3% (0.9)	91.3% (1.6)	80.7% (2.7)	79.0% (3.1)	80.7% (2.7)	79.2% (3.0)	80.6% (2.2)	79.2% (3.0)	75.9% (3.4)	76.2% (3.6)	75.9% (3.4)
Sonar	87.7% (1.2)	84.2% (1.4)	74.5% (2.7)	74.5% (2.7)	74.5% (2.7)	73.0% (2.7)	73.0% (2.7)	73.0% (2.7)	70.3% (2.5)	70.3% (2.5)	70.3% (2.5)
Vote	91.6% (0.8)	88.8% (0.7)	88.2% (1.1)	87.2% (1.6)	88.2% (1.1)	89.1% (1.0)	87.5% (1.5)	89.1% (0.9)	87.1% (1.2)	87.1% (1.5)	87.0% (1.2)
Vote+WoE	91.6% (0.8)	88.8% (0.9)	88.4% (1.4)	87.3% (1.3)	88.4% (1.4)	89.0% (1.0)	87.5% (1.7)	89.0% (1.0)	86.8% (1.3)	87.0% (1.3)	86.8% (1.3)
Wine	98.9% (0.5)	97.9% (0.4)	92.3% (1.6)	91.9% (1.7)	92.3% (1.6)	93.1% (1.2)	92.7% (1.0)	93.1% (1.2)	90.0% (2.4)	90.2% (2.4)	90.0% (2.4)

with RF predictions has little effect, mainly because the RF often achieves 100% training accuracy. Replacing the labels with SVM predictions slightly affects white-box accuracies, however it does not always lead to improved performance.

It was experimentally compared how the accuracy of the black-box learner varies when the data set is extended with additional samples generated using the following methods. The one letter abbreviation are used in tables and plot legends.

1. Independent uniform sampling (abbrev. u) over the range of attributes. For discrete attributes all possible values are sampled with the same probability.

2. Independent sampling of values for different attributes from univariate FKDEs. Results with no smoothing (kernel width $h = 0$) (abbrev. d) are compared with results with the Gaussian kernel of width determined using equation (39) (abbrev. k). Nominal attributes are sampled independently according to their empirical marginal distributions.

3. Sampling from a multivariate FKDE. Numerical and nominal attributes are treated separately. For numerical ones the results when the data are sphered and the Gaussian kernel width is given by equation (44) (abbrev. f) are compared with results that use a product of Gaussian kernels of width determined using equation (46) (abbrev. m). For nominal attributes the kernel (49) is used with λ set using equation (51).

4. Sampling from Weka's "EM" clustering (abbrev. e). Numerical attributes are modeled as multivariate Gaussians, nominal ones are assumed to be conditionally independent given the cluster and are specified using their conditional marginal probabilities.

5. Independent sampling of features in the ICA space (abbrev. i). Independent components are learned using the FastICA algorithm [113] and independent univariate FKDEs are used to estimate the density of the data in the feature-space. This method handles only numerical attributes.

6. Multivariate sampling of features in the PCA space (abbrev. p). The density of the data in the feature space is estimated using FKDE with a product kernel because PCA features are not correlated. This method handles only numerical attributes.

7. Sampling using the ALBA method (abbrev. s) [65]. It corresponds to sampling from an independent univariate KDE of the density of support vectors of an SVM. This method handles only numerical attributes.

For every data set, black-box, and white-box method combination the CV accuracy for increased quantities of artificially generated samples was recorded. The number of artificial samples was limited to 10000 and the size of the extended data is reported in multiples of the original data size. Thus relative size 1 means that only the original data is used. To reduce the variability of results all white-box classifiers were trained on exactly the same augmented data sets, and the same instances of black-box classifiers were used to label data generated with all evaluated methods. This implementation of the experiment allowed testing how samples generated by the SVM-dependent ALBA perform when the RF is used to obtain the labels.

The most informative way to present the results is to graph the accuracy of white-box learners against the size of the extended training data sets. It is also informative to analyze how the accuracy varies with the size of the white-box classifiers. Such plots are shown for the data set "Balance Scale" in Figure 9. In all accuracy plots the solid horizontal line at the top presents the cross-validation accuracy of the SVM, while the dashed horizontal line near the bottom depicts the accuracy of the JRip rule learner on the original data. Accuracies of the rule learner obtained on extended datasets are plotted between those two reference levels in function of the relative data set size (the size of the extended dataset divided by the original size, 1 means that only the original samples were used). Due to the large number of possible combinations of white-box, black-box, and sampling method a few of the graphs were selected for further discussion in the text. To summarize the graphs the results were tabulated using the following measures. Maximums over all

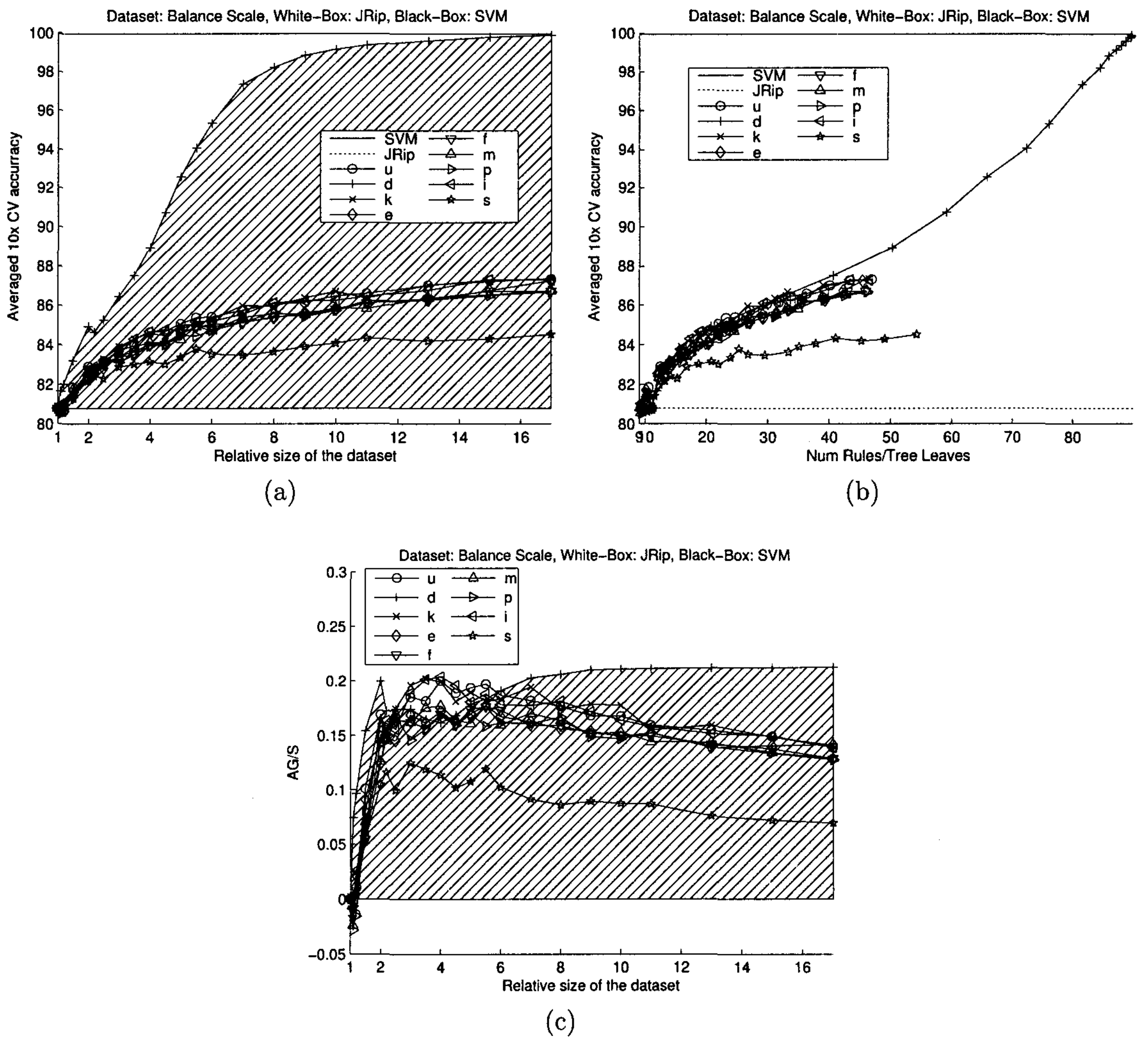


Figure 9: Results of experiments on the "Balance Scale" data with JRip rules learned from samples labeled using an SVM. 10xCV is plotted as a function of the increase of training data set (a) and as a function of the number of rules (b). Accuracy Gain divided by rule set Size AG/S as a function of the increase of training data (c). Areas of the marked regions are reported in Tables 4 and 5 as AG and AG/S.

training set sizes of the averaged CV accuracies are reported in Table 3. A useful figure of merit is the Accuracy Gain (AG) defined to be the difference between the accuracy of the white-box learner on the augmented and original data sets. For example, the AG when independent sampling from marginal distributions (d) was used to generate new data is indicated in Figure 9a. In addition, to analyze the trade-off between white-box accuracy and comprehensibility the accuracy gains were divided by the size of the white-box model (AG/S), which is the number of rules for JRip and number of leaves for J48. To capture the variability of AG and AG/S with the relative size of the data their integrals were computed over the relative data set sizes. Those are reported in Tables 4 and 5. They correspond to the areas of the shaded regions in Figure 9a and Figure 9c. Since the accuracy gain depends on the white-box original performance, the values in Tables 4 and 5 can be compared between sample generation methods and black-box architectures for a selected data set and white-box architecture combination. In general, good methods demonstrate both large AG and large AG/S integrals.

The largest increase of the white-box accuracy is observed on the "Balance Scale" data set on which independent sampling from non-smoothed marginal distributions matches the accuracy of the SVM. It has already been shown in Figure 9 how the accuracy of JRip varies with the amount of training data and with the number of rules. The results on "Balance Scale" must be, however, taken with a grain of salt because the data is fairly atypical: there are only four numerical attributes that take only five different values each, which limits the attribute space to just 625 points. Since the data space is small, the independent generator is able to enumerate all points. The graphs match the two summary values in Tables 4 and 5. The un-smoothed independent FKDE generator (d) has demonstrated the largest gain in accuracy and the best ratio of accuracy gain to the number of rules. Despite the atypical nature of the "Balance Scale," the results demonstrate an important property of the non-smoothing FKDE generator. It is robust to data peculiarities, such as numerical attributes that nevertheless take only a small number of values. All other density estimation methods lead to lower accuracies because they fail to

TABLE 3: Maximum of averages of 10xCV accuracies. For each data set indicated are in each column the values that are within 1% of the maximum.

		JRip Acc		J48 Acc		J48+REP Acc	
		SVM	RF	SVM	RF	SVM	RF
Bal	u	87.3	**82.9**	87.8	**81.0**	87.0	**80.7**
	d	**99.9**	82.4	**99.9**	**81.2**	**99.8**	**81.1**
	k	87.3	**83.4**	87.4	**81.0**	86.9	**80.8**
	e	87.2	**83.6**	86.3	**80.9**	86.2	**80.7**
	f	86.7	**83.2**	86.0	**81.0**	85.8	**80.7**
	m	86.7	**83.5**	86.1	**81.0**	85.9	**80.6**
	p	86.7	**83.5**	86.2	**81.0**	85.9	**80.6**
	i	87.3	**83.4**	87.2	**80.9**	86.8	**80.8**
	s	84.5	80.8	82.8	78.0	82.1	78.2
Ger+WoE	u	**74.5**	74.4	74.3	72.6	74.0	74.1
	d	**74.8**	**75.9**	**75.1**	**75.5**	**74.8**	**75.4**
	k	**74.8**	**75.8**	**74.6**	**75.0**	**74.5**	**75.3**
	e	**74.4**	**75.2**	74.3	74.0	73.6	**74.8**
	f	**74.5**	75.1	**74.6**	74.0	**74.1**	**74.8**
	m	**74.6**	**75.3**	**74.5**	74.0	**74.1**	74.4
	p	**74.7**	**75.3**	74.3	74.1	**74.1**	**74.9**
	i	**74.6**	75.0	**74.4**	73.9	**74.1**	**74.8**
	s	**74.7**	73.4	74.3	72.2	**74.1**	72.5
Ger	u	72.0	72.9	**73.3**	72.9	**73.2**	72.4
	d	**72.6**	**73.5**	**73.1**	**74.2**	**73.2**	**73.7**
	e	**72.5**	**73.0**	**73.2**	**74.0**	**73.2**	73.6
	f	**73.1**	**73.7**	**73.6**	**74.6**	**73.6**	**74.4**
Prom+WoE	u	85.7	87.7	83.4	85.3	81.4	84.8
	d	**89.1**	**90.9**	84.8	**89.7**	87.8	**90.9**
	k	87.7	**90.7**	85.0	89.2	86.8	**91.5**
	e	86.3	87.4	86.0	87.4	83.5	85.9
	f	**89.9**	**90.3**	**88.2**	**89.6**	**88.6**	**91.1**
	m	87.4	88.9	84.9	87.6	85.4	88.0
	p	**89.5**	**91.2**	**88.5**	**90.5**	**89.5**	**90.8**
	i	**89.2**	**91.0**	86.4	89.5	**88.8**	**91.5**
	s	83.9	83.8	84.2	84.1	83.7	83.6
Prom	u	80.7	86.8	80.6	83.2	77.7	83.3
	d	**86.6**	86.3	**83.5**	88.1	84.2	**88.6**
	e	85.7	**88.2**	**83.4**	**89.0**	84.3	**88.6**
	f	**87.0**	86.6	**83.8**	**88.4**	**85.3**	**88.8**

Continued on next page...

TABLE 3 (continued): Maximum of averages of 10xCV accuracies.

		JRip Acc		J48 Acc		J48+REP Acc	
		SVM	RF	SVM	RF	SVM	RF
Sonar	u	75.3	74.5	73.8	75.0	70.8	71.7
	d	75.1	76.0	73.4	75.5	71.5	74.1
	k	74.5	75.9	74.8	75.2	71.3	74.3
	e	75.3	75.4	74.6	76.0	72.3	74.2
	f	**82.3**	**79.8**	**80.1**	**78.6**	**80.5**	**77.8**
	m	**81.7**	**79.3**	**80.0**	**78.0**	79.6	76.9
	p	**82.1**	**79.7**	**80.5**	**78.2**	79.6	77.0
	i	80.3	77.5	77.3	76.7	76.8	75.3
	s	77.1	78.1	74.1	74.3	72.3	72.3
Vote	u	**90.6**	**89.3**	**90.3**	**89.1**	**90.0**	**89.3**
	d	**90.8**	**89.5**	**90.6**	**89.3**	**90.5**	**89.4**
	e	**90.9**	**89.3**	**90.5**	**89.2**	**90.5**	**89.2**
	f	**91.0**	**89.1**	**90.9**	**89.1**	**90.6**	**89.2**
Vote+WoE	u	89.7	**89.4**	88.8	**89.0**	88.7	**89.2**
	d	**90.9**	**89.4**	**90.6**	**89.2**	**90.5**	**89.4**
	k	89.9	**89.6**	**90.2**	**89.0**	**89.9**	**89.2**
	e	89.6	**89.4**	**90.0**	**89.2**	**89.8**	**89.1**
	f	89.8	**88.9**	**89.8**	**89.0**	89.5	88.4
	m	89.6	**88.9**	**90.0**	**89.0**	**89.6**	**88.7**
	p	89.7	**88.8**	**89.7**	**89.2**	**89.7**	88.4
	i	89.6	**89.3**	**90.2**	**89.0**	**89.7**	**89.2**
	s	87.9	88.4	88.1	**89.0**	87.4	87.4
Wine	u	94.6	**96.8**	94.1	**96.6**	92.9	**96.4**
	d	**96.5**	**97.0**	95.2	**97.0**	94.6	**96.5**
	k	96.1	**97.1**	95.3	**96.8**	94.5	**96.5**
	e	95.9	96.2	94.9	95.9	94.4	95.2
	f	**97.2**	**97.2**	**96.4**	**97.4**	**96.5**	**97.1**
	m	**96.8**	**97.1**	**96.3**	**97.1**	**95.9**	**96.9**
	p	**96.9**	**97.3**	**96.7**	**97.3**	**96.3**	**96.9**
	i	**96.8**	**97.1**	**96.1**	**97.4**	**96.1**	**96.9**
	s	94.0	95.9	**96.4**	**96.9**	**96.0**	**96.3**

TABLE 4: Summary of experiments using JRip white-box.
$\int AG$ is the integral of Accuracy Gain (the difference between the white-box's accuracy on expanded and original data) over relative set sizes. $\int AG/S$ is the integral of the Accuracy Gain divided by the Size of the classifier (number of rules or tree leaves). For each column and each dataset indicated are in bold the values that are within 5% of the maximum.

		JRip			
		SVM		RF	
		∫AG	∫AG/S	∫AG	∫AG/S
Bal	u	79.3	2.595	26.0	0.949
	d	**237.2**	**3.135**	-0.4	0.032
	k	79.2	2.619	28.5	1.071
	e	71.6	2.369	**32.8**	**1.209**
	f	70.8	2.333	**31.4**	1.136
	m	70.6	2.343	**32.4**	**1.182**
	p	69.7	2.310	**31.3**	1.134
	i	77.9	2.567	29.3	1.102
	s	46.0	1.390	-38.3	-1.248
Ger+WoE	u	10.2	0.516	7.9	0.580
	d	**13.4**	0.744	**21.8**	1.433
	k	12.5	**0.822**	20.2	**1.676**
	e	8.4	0.541	13.1	1.061
	f	10.9	0.629	13.6	0.948
	m	11.1	0.642	13.9	0.973
	p	11.3	0.664	14.4	1.020
	i	10.9	0.622	12.8	0.942
	s	12.0	0.539	-8.5	-0.307
Ger	u	-16.7	-1.199	4.0	0.395
	d	-0.0	-0.105	**7.6**	0.417
	e	-0.1	-0.042	3.2	0.200
	f	**3.8**	**0.245**	**7.9**	**0.530**
Prom+WoE	u	134.6	19.809	243.7	**33.611**
	d	237.5	10.178	**354.4**	21.818
	k	180.9	12.312	338.6	**32.659**
	e	166.8	18.682	228.1	26.669
	f	**304.1**	**22.017**	**353.4**	31.129
	m	205.1	14.184	267.9	20.840
	p	278.2	20.347	**348.6**	30.559
	i	267.7	18.855	**358.4**	31.198
	s	-14.2	-2.436	-4.4	-0.647

Continued on next page...

TABLE 4 (continued): Summary of experiments using JRip white-box.

		JRip			
		SVM		RF	
		∫AG	∫AG/S	∫AG	∫AG/S
Prom	u	-127.2	-8.836	188.8	**12.913**
	d	**162.1**	**6.508**	168.5	8.457
	e	141.5	5.641	**228.1**	12.037
	f	**154.3**	5.498	195.1	9.622
Sonar	u	-90.4	-6.907	-99.7	-16.794
	d	-19.2	-1.618	45.8	1.965
	k	-73.3	-4.786	32.2	1.399
	e	-3.8	-0.643	15.5	1.000
	f	**298.8**	**13.314**	177.3	8.733
	m	255.3	10.976	**180.3**	8.635
	p	279.5	12.470	**189.1**	**9.213**
	i	207.8	9.328	93.6	4.549
	s	43.5	3.279	79.7	5.303
Vote	u	34.9	1.556	12.4	0.812
	d	40.0	1.723	**20.1**	**1.245**
	e	**41.3**	**2.996**	13.5	0.844
	f	**42.6**	2.729	11.8	0.659
Vote+WoE	u	15.3	1.203	11.3	1.334
	d	**33.9**	**1.570**	**16.3**	1.111
	k	21.7	1.413	**15.9**	**1.751**
	e	16.2	1.268	13.9	**1.803**
	f	15.1	0.975	-10.3	-0.963
	m	12.7	0.852	6.2	0.692
	p	14.7	0.994	-4.4	-0.422
	i	17.6	1.291	11.1	1.435
	s	-22.5	-2.065	-97.6	-7.314
Wine	u	77.0	2.117	184.8	9.946
	d	156.6	4.185	196.3	8.445
	k	147.8	4.983	**197.8**	10.629
	e	121.3	8.742	152.9	12.995
	f	**204.9**	**9.788**	**204.6**	12.907
	m	178.9	7.637	**204.5**	11.594
	p	190.7	8.650	**207.6**	12.666
	i	186.6	7.374	**203.4**	11.755
	s	63.8	6.296	152.2	**14.193**

TABLE 5: Summary of experiments using J48 white-box.
$\int AG$ is the integral of Accuracy Gain (the difference between the white-box's accuracy on expanded and original data) over relative set sizes. $\int AG/S$ is the integral of the Accuracy Gain divided by the Size of the classifier (number of rules or tree leaves). For each column and each dataset indicated are in bold the values that are within 5% of the maximum.

		J48				J48+REP			
		SVM		RF		SVM		RF	
		∫AG	∫AG/S	∫AG	∫AG/S	∫AG	∫AG/S	∫AG	∫AG/S
Bal	u	115.4	0.542	39.2	0.272	106.4	1.168	29.3	0.282
	d	**304.4**	**1.997**	**47.6**	**0.348**	**266.1**	**1.928**	**38.9**	**0.315**
	k	112.8	0.564	39.1	0.266	106.4	1.237	31.1	**0.319**
	e	101.1	0.503	39.1	0.265	96.8	1.105	30.4	0.293
	f	99.0	0.504	36.8	0.251	92.1	1.073	27.8	0.275
	m	98.8	0.505	38.7	0.263	93.8	1.097	28.9	0.284
	p	98.9	0.509	38.2	0.260	94.4	1.094	29.1	0.283
	i	109.4	0.549	39.3	0.270	104.5	1.223	31.4	**0.313**
	s	69.5	0.198	-43.2	-0.503	53.2	0.348	-99.9	-1.689
Ger+WoE	u	19.6	0.097	2.2	0.010	15.8	0.211	13.8	0.223
	d	**25.2**	**0.167**	**28.6**	**0.155**	**20.4**	0.273	**26.5**	0.321
	k	21.8	0.147	19.7	0.102	**19.7**	**0.321**	21.7	0.336
	e	18.7	0.118	13.9	0.078	12.5	0.222	18.3	**0.368**
	f	20.5	0.136	13.2	0.067	16.3	0.269	18.6	0.316
	m	20.0	0.132	14.0	0.066	16.1	0.228	15.9	0.201
	p	19.3	0.125	12.2	0.064	14.8	0.243	19.5	0.338
	i	19.6	0.127	13.7	0.076	15.7	0.263	18.1	0.342
	s	18.1	0.135	-8.5	-0.028	15.3	0.216	-7.3	-0.044
Ger	u	3.5	0.019	13.7	0.060	-11.0	-0.030	1.8	0.005
	d	11.6	0.043	26.8	**0.125**	0.9	0.009	14.0	**0.082**
	e	14.1	0.049	24.9	0.113	2.9	0.015	12.4	0.067
	f	**22.0**	**0.072**	**29.5**	0.116	**11.7**	**0.046**	**16.8**	0.077
Prom+WoE	u	224.2	3.119	339.6	5.536	119.3	8.490	295.8	17.094
	d	283.0	1.508	**540.1**	4.386	419.1	7.845	**623.1**	17.610
	k	317.3	2.487	538.2	5.230	360.7	8.399	**619.7**	22.510
	e	372.2	8.092	449.7	10.713	274.0	16.895	370.0	22.328
	f	**482.4**	6.928	**556.6**	9.687	**496.7**	18.416	572.6	24.534
	m	333.5	5.253	441.2	7.140	363.0	13.277	457.8	17.029
	p	**476.6**	6.547	**568.2**	9.473	**508.2**	18.787	**603.1**	26.157
	i	410.6	5.309	539.4	8.332	**483.9**	16.965	**620.0**	26.966
	s	301.8	**29.456**	308.3	**31.810**	279.6	**30.755**	287.1	**31.844**

Continued on next page...

TABLE 5 (continued): Summary of experiments using J48 white-box.

		J48				J48+REP			
		SVM		RF		SVM		RF	
		∫AG	∫AG/S	∫AG	∫AG/S	∫AG	∫AG/S	∫AG	∫AG/S
Prom	u	-392.7	-8.801	126.7	1.042	-294.7	-8.792	260.0	5.009
	d	97.4	0.025	328.6	1.394	270.0	2.297	**496.6**	**5.491**
	e	68.1	-0.106	**350.5**	**1.612**	252.8	2.194	482.8	**5.540**
	f	**110.6**	**0.149**	322.9	1.463	**298.3**	**2.594**	**509.1**	**5.777**
Sonar	u	-87.3	-0.353	-47.7	-0.525	-135.0	-2.682	-65.4	-4.775
	d	-99.5	-0.304	78.4	0.306	-45.2	-0.399	139.8	2.547
	k	-85.7	-0.205	65.1	0.283	-63.7	-0.636	152.5	2.707
	e	-36.9	-0.038	108.0	0.670	43.8	0.886	141.0	3.101
	f	**290.6**	**1.466**	**222.9**	**1.631**	**409.3**	**5.714**	**280.7**	**5.830**
	m	271.8	1.184	200.7	1.123	350.0	4.529	248.6	4.225
	p	**284.2**	**1.418**	209.4	**1.591**	384.4	5.414	**272.4**	**5.669**
	i	167.3	0.890	141.5	1.088	267.8	4.095	199.4	4.346
	s	-97.1	-2.010	-103.4	-1.512	-21.9	-0.275	-15.5	-0.084
Vote	u	17.4	0.171	-5.2	-0.159	48.1	1.107	36.4	0.993
	d	22.4	0.212	-2.0	-0.089	57.2	1.201	**38.3**	1.006
	e	20.8	0.504	-1.3	-0.123	58.0	**2.581**	34.9	**1.268**
	f	**26.5**	**0.537**	-3.3	-0.158	**61.9**	2.101	34.7	0.975
Vote+WoE	u	-11.1	-0.232	-9.3	-0.235	26.5	1.016	38.6	1.750
	d	**24.8**	**0.250**	**2.3**	-0.003	**61.6**	1.315	**46.0**	1.202
	k	13.5	0.106	-8.1	-0.201	54.4	1.768	38.0	1.559
	e	11.1	0.103	-13.3	-0.386	45.3	**2.114**	38.9	2.081
	f	4.1	-0.030	-24.2	-0.706	40.8	1.743	17.5	1.093
	m	9.7	0.048	-20.0	-0.525	44.4	1.842	25.3	1.225
	p	6.0	-0.033	-22.6	-0.674	41.2	1.702	24.6	1.378
	i	13.2	0.107	-12.5	-0.380	45.7	1.892	37.7	**2.428**
	s	-32.2	-0.800	-125.8	-3.167	-2.4	-0.046	-102.1	-3.066
Wine	u	20.5	0.007	154.0	1.389	110.3	1.508	286.9	6.947
	d	60.5	0.110	162.2	1.435	157.8	1.904	**288.5**	6.826
	k	61.1	0.083	159.5	1.400	158.6	1.811	282.4	6.745
	e	45.4	0.415	81.9	1.651	159.4	6.605	194.0	9.063
	f	**133.7**	1.329	**172.8**	2.501	**268.8**	6.705	**293.8**	9.472
	m	120.7	1.014	167.3	2.085	247.6	5.360	**297.1**	8.147
	p	**132.9**	1.094	**170.0**	2.330	265.0	6.115	**294.8**	9.303
	i	114.0	0.790	**177.5**	2.139	251.8	4.822	**302.4**	8.797
	s	**135.4**	**6.044**	155.9	**5.532**	**279.2**	**17.630**	**292.1**	**17.020**

preserve this characteristic of the data.

The results obtained on the "Wine" data set are more typical and prevail in most other cases. Similarly to the "Balance Scale" data, Figure 10 presents the relationships between the accuracy of the J48 classifier and the number of generated samples and tree size. On this data set, when the SVM is used to provide labels, the ALBA method (s) generates samples that result in both the smallest and most accurate trees. Moreover the accuracy versus tree size plot reveals three groups of curves. Samples generated by the ALBA method quickly lead to concise and accurate trees. It was found that the ALBA method often leads to small classifiers, however they are seldom as accurate as in this case. In the second group are methods that model attribute correlations. Multivariate FKDE (f, m), ICA-based (i), and PCA-based (p) estimators yield a similarly high accuracy, but the trees are larger. Finally, the three methods that assume attribute independence – smoothed (k) and un-smoothed (d) FKDE and uniform sampling (u) lead to the least accurate and largest trees.

Careful inspection of the results on the "Wine" data set reveals another important characteristic of the proposed rule extraction method. Even though, as seen from Table 2, the Random Forest is 1% less accurate than the SVM, decision trees on additional data labeled by the RF are more accurate. This is visible in Figure 11. Despite averaging 25 runs of experiments, the curves show a large degree of variability. Therefore it is better to refer to the accuracy gains from Tables 4 and 5 instead of the top accuracies in Table 3, which has been brought for convenience and reference only. When the labels are produced with the SVM the best sample generator is ALBA (s), with the AG integral of 135.4. On the same samples, but relabeled using the RF, ALBA's AG integral raises to 155.9, without increasing considerably the tree size. However, it is surpassed by other methods: ICA-based (i) generator has the maximum AG of 177.5, followed by FKDE (f) generator with AG=172.8, and PCA-based generator (p) with AG=170.

The "Promoter" data set to illustrates the impact of the two possibilities of

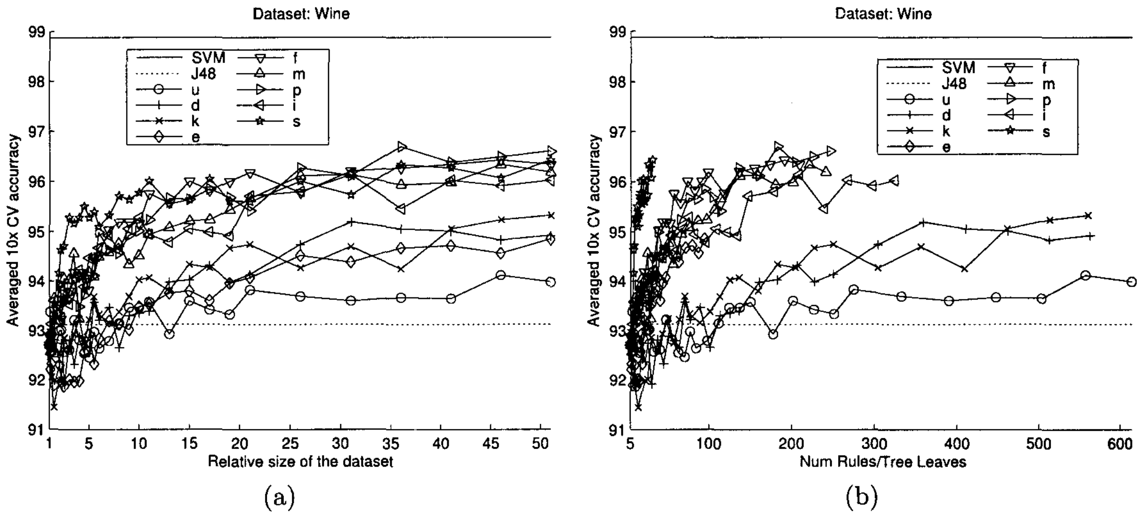


Figure 10: Results on the "Wine" data set with J48 trees learned on samples labeled using an SVM. 10xCV is plotted as a function of the increase of training data set (a) and as a function of the number of leaves (b).

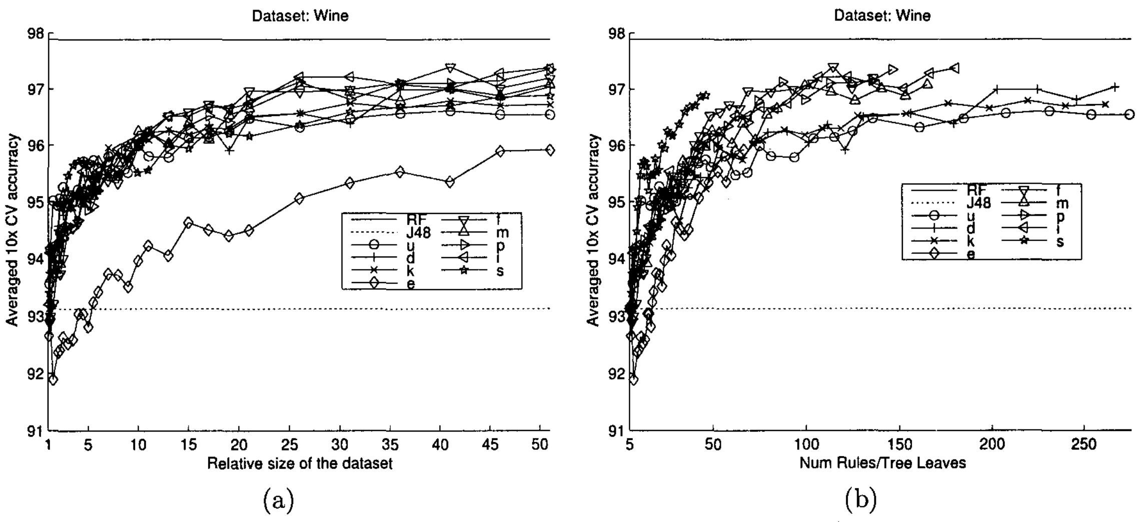


Figure 11: Results on the "Wine" data set with J48 trees learned on samples labeled using a RF. 10xCV is plotted as a function of the increase of training data set (a) and as a function of the number of leaves (b).

handling discrete attributes. First, the original data with nominal attributes is used. The density estimation methods that can be used are uniform sampling (u), independent sampling of marginal distributions (the smoothed estimator (k) and un-smoothed estimator (d) are equivalent), multivariate kernel density estimate (the sphering estimator (f) and the product kernel estimator (m) are equivalent), and a EM mixture model (e). The results of J48 with reduced error pruning on data labeled with a RF have been graphed in Figure 12. EM and the two kernel density estimators yield very comparable accuracy. Uniform sampling is clearly inferior.

Alternatively, nominal attributes can be transformed into numerical ones using the Weight of Evidence method. It makes it possible to apply all density estimation techniques. The results are presented in Figure 13. The first observation is that after the WoE transformation both the black-box and the white-box learners demonstrate increased accuracies. It may be due to the fact that the task becomes easier because the WoE transform is applied before to the whole training data before it is split for cross-validation. Independent marginal density estimators (d, k) and multivariate estimators (f, p, i) result in similar high accuracy, which agrees with the results on the original, unprocessed data. However, see that the accuracy produced on samples generated from the Gaussian Mixture Model (e) has deteriorated. Also, the accuracy obtained using the multivariate FKDE with a product kernel (m) is significantly worse than when data sphering is used to select kernel width (f).

C Recommendations

The reported results demonstrate that the simple algorithm presented in Algorithm 1 consistently improves the accuracy of the understandable learners. However, the extent of improvement heavily depends on the choice of method used to estimate the probability density of new data. The experimental results allow to formulate the following conclusions.

The study shows that efficiency of the rule extraction as learning approach depends on both the quality of the black-box and the density estimation method. As described in Chapter II, both SLT and PAC learning theories assume that the

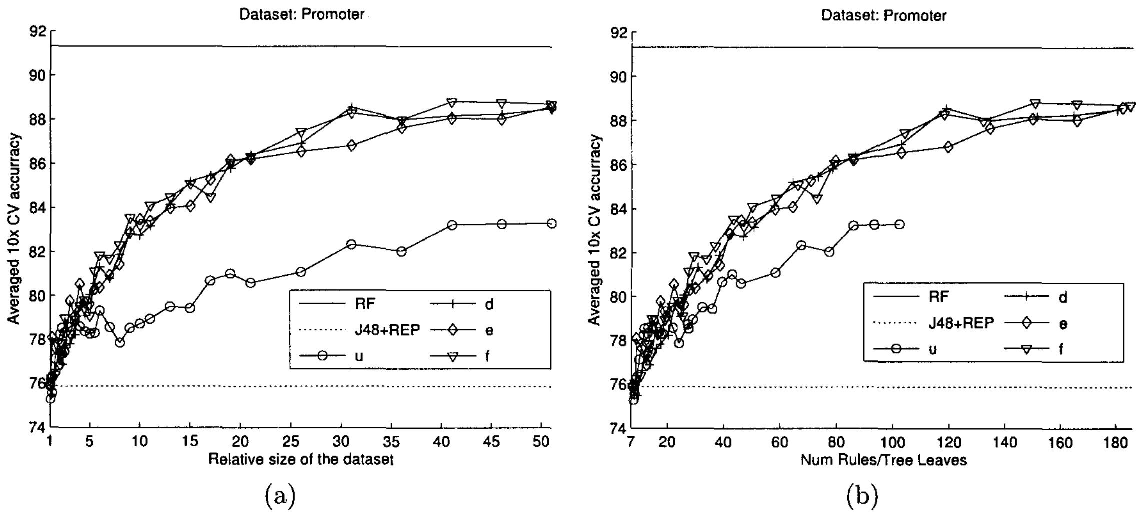


Figure 12: Results on the "Promoter" data set with J48 with REP trees learned on samples labeled using a RF. 10xCV is plotted as a function of the increase of training data set (a) and as a function of the number of leaves (b).

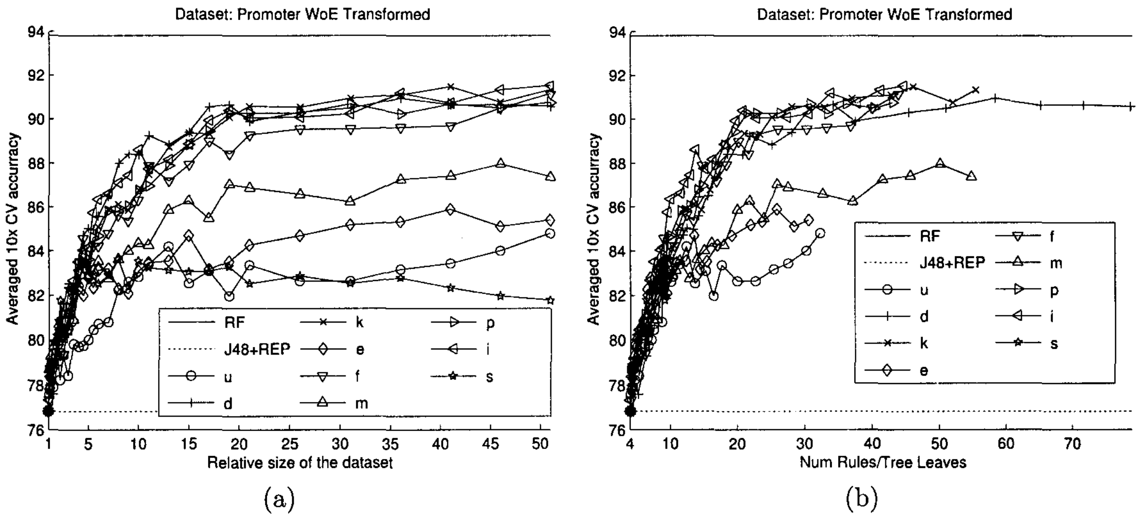


Figure 13: Results on the "Promoter" data with nominal attributes replaced by their weight of evidence. J48 with REP trees are learned on samples labeled using a RF. 10xCV is plotted as a function of the increase of training data set (a) and as a function of the number of leaves (b).

training and testing data come from the same probability distribution. Artificial data is generated according to equation (35). Its consistency with the underlying but unknown data probability density depends on both the quality of the black-box and on the density estimator.

The rule extraction process strongly depends on the accuracy of the black-box. The assumption that the black-box outperforms the white-box, or in other words that there is an accurate classifier available is crucial to the rule extraction process. Often, when the black-box generalization ability is worse than that of the white-box, the final accuracy deteriorates. This can be seen for the combination of J48 and Random Forest on the "Vote" data where the single decision tree outperforms the forest and the resulting accuracy gains are negative.

To ensure optimal generalization capability and robustness of the black-box classifier different architectures should be evaluated. However, the best-performing black-box may not lead to the best white-box accuracies. The results indicate that white-boxes trained on data relabeled by Random Forests were smaller and more accurate than those trained on data relabeled by Support Vector Machines, even though SVMs had higher accuracy than Random Forests. A possible explanation is that it is beneficial to match the inductive biases of the white-box and black-box classifiers.

The theoretical bounds on the generalization error also require that the attributes are generated from the real, underlying density. Under mild conditions, that are satisfied for the Gaussian kernel and the width selection rule (39), the FKDE estimate is asymptotically consistent with the unknown underlying probability distribution [108]. This means that when the number of available samples tends to infinity, the discrepancy between the actual and estimated probability distribution approaches zero. This fact motivates the use of kernel density estimators.

The importance of matching the distribution of the generated samples to the underlying data distribution can be justified by a comparison of two naive estimators: the uniform one and the univariate kernel density estimator. On the majority of the datasets used in the experiments, the kernel estimators (k,d) outperform the uniform estimator (u). A possible explanation is that the uniform generator results in lower accuracy, because it does not preserve the changes in the density of the data, which may lead to class imbalances and over-expression of regions where the data were scarce.

On the other hand, experiments also shown that even an approximate match of the estimate to the underlying data distribution can often result in good accuracy. Often the accuracy obtained under attribute independence assumption and without it are very similar. This may be justified by the fact, that if the artificial data covers regions where the original data are improbable, the testing set will not contain samples that belong to those regions. Thus the accuracy will not be affected. However, the white-box classifier must have terms that determine its decision boundary for those spurious regions of the attribute space, its understandability will thus deteriorate. This justifies the experimentally observed results, in which modeling of attribute inter-relations resulted mainly in smaller white-boxes that were similarly accurate to those built under the initial assumption of attribute independence.

If attribute independence is assumed and the correlations are not taken into the account, then the univariate KDEs have proved to be a good choice. The un-smoothed estimator with zero-kernel width (d) often yields high accuracy and has the advantage of not generating artificial attribute values. This characteristic has been exemplified on the "Balance Scale" data set. The case of a numerical attribute that takes only discrete values is not uncommon. In fact, such attributes are created by the WoE transformation. On the other hand, the smoothed estimators (k) often yielded slightly smaller white-box classifiers.

When a multivariate density estimation is required, then the multivariate FKDE of sphered data (f) is a good first choice. On all tested data it has resulted in accuracies comparable to other methods that model attribute inter-dependencies. Moreover the use of data sphering (d) instead of product kernels (m) has led to better accuracies on all data sets on which the multivariate estimates resulted in better accuracies that independent univariate estimates.

The proposed kernel for nominal data (49) performs very well. It induces a sampling algorithm that is very similar to the mutation operator commonly used in genetic optimization. It is very flexible and for $\lambda = 0$, it is equivalent to the independent sampling from attributes' marginal distributions, while when $\lambda = 1$ only copies of samples already belonging to the data set will be generated. Currently, the parameter λ is set by maximizing the problem (51). However, the computation of the objective has a quadratic runtime complexity on the number of training samples. More efficient methods of estimation of the parameter λ are needed.

Biasing the data density estimate toward the black-box decision boundary, in the spirit of the ALBA method usually yields very concise white-box classifiers. However, it often yields smaller accuracy gains than the multivariate kernel density estimation methods. Moreover, in extreme cases it may cause the accuracy of the white-box classifier to deteriorate (e.g. on the "Sonar" data set with decision tree learners, or on the WoE transformed sets "Promoter" and "Vote"). Further research is needed to verify whether this behavior is due to the attribute independence assumption made in the ALBA method, or to the closeness of generated samples to SVM's decision boundary.

Finally, the study points out that it is important to choose a good white-box learner. The experiments suggest the use of RIPPER rule learner rather than the C4.5 decision tree algorithm, because the accuracies are often very similar and RIPPER rule sets tend to be much smaller. This behavior was attributed to the rule pruning strategy used by RIPPER, which grows and prunes rules on separate data and enforces a model description length limit for the whole rule set [14]. To evaluate how a similar pruning strategy affects decision trees, the J48 learner was trained with reduced error pruning (REP) enabled.

J48 decision trees learned with REP enabled had smaller sizes without significant changes of their accuracies. This behavior can be motivated as follows. The original pessimistic confidence interval pruning heuristic was designed to use all available data to build the tree. As a consequence, the pruning decision is based on a single statistical test whose assumptions are not fully satisfied [13]. On the other hand, reduced error pruning divides the training data into a growing and pruning set, which allows it to make more reliable decisions about node pruning. Artificial data can be generated, therefore it becomes more important to reliably prune the tree, rather than use all data to build the tree.

D Conclusions

Several methods of probability density estimation were described and evaluated in the context of improving the accuracy of understandable classifiers. The proposed method is applicable to any combination of black-box and white-box classifiers and can handle both numerical and nominal data, which is an important practical issue. Experimental results allow the formulation of recommendations about applying the presented methods in practical situations. It is believed that similarly to commonly used data preprocessing and filtering techniques, the sample generation methods outlined and evaluated in this contribution may become essential building-blocks of bigger data-mining projects.

CHAPTER IV

INDUCING RODDS FROM NEURAL NETWORKS

In a practical setup, the rule extraction algorithm must internally encode the rules it finds into a data structure. On the one hand, this structure should allow easy manipulation of the rules. If the internal rule representation is not the one used for rule display and presentation to the user it should also be able to concisely express all the features of the presentation language. On the other hand, in the context of rule extraction the internal data structure should make it easy to encode concepts that are natural for the analyzed black-box classifier. The Reduced Ordered Binary Decision Diagrams (ROBDDs) and their extensions presented in this chapter fulfill many of these requirements.

ROBDDs were proposed by Bryant as an efficient data structure for the manipulation of boolean functions [115]. They have been extensively used in the design of digital integrated circuits. Many extensions have been proposed since, to enhance their capabilities [116–120], and they remain the subject of continuous research. The discovery and adoption of ROBDDs "have led to dramatic performance improvements and breakthrough in many CAD projects all over the world" [121]. They offer the means to represent and process boolean functions defined over thousands of variables, which open the door to verification, optimization and implementation of complicated integrated circuit structures. Reduced Ordered Decision Diagrams (RODDs) are a generalization of ROBDDs for functions taking attributes defined over finite domains.

A Main Properties of RODDs

The RODDs demonstrate excellent algorithmic properties for function manipulation over finite domains:

- They provide a canonical representation of functions i.e. every function has a unique representation. This leads to easy testing for equivalence, tautology, satisfiability, and falsifiability.
- The complexity of performing any boolean operation on two functions given as RODDs is proportional to the product of their sizes (the number of nodes in the diagrams).
- Counting the number of satisfying assignments for a function and finding the next such assignment has a complexity proportional to the size of the function representation.
- Many practically used functions have small RODD representations, especially all symmetric functions (e.g. parity, M-of-N conditions, equality, and inequality).

The downsides of RODDs are that some functions have representations that require an exponential number of nodes (most notably the function for the middle bits of multiplier circuits). Furthermore, a sequence of operations on RODDs may run in an exponentially growing time, because at each operation the result may grow as the product of sizes of the operands. RODDs are also highly sensitive to the order of variables (see Section 2).

Small size of RODD representation of processed functions is crucial, as the performance of operations depends on it. Therefore, a lot of research in VLSI design has been targeted to extend the RODD methodology to provide smaller representations for many classes of functions, at the cost of the increased complexity of some of listed operations.

1 RODDs in Machine Learning

There exist multiple similarities between the goals encountered in ML and VLSI design [122]. A substantial effort has been made in ML to find classifiers of minimal complexity because, according to Occam's razor, a simple expression of a concept will be more general and will better classify unknown samples. Consequently, ML methods have been used for boolean function minimization [123]. On the other hand, minimal implementations are vital for efficient design of integrated circuits to save chip space. To this end minimizing ROBDDs of only partially specified functions (with don't cares) has been studied [124–127]. Note, that this corresponds to learning a classifier compatible with the data set, i.e. a function from the input space into class labels whose value is initially specified only on the data set. A fundamental difference between the VLSI and ML task is, however, that in VLSI design the function is undefined on a small part of the attribute space, whereas in a realistic ML setup it is undefined on most of the attribute space. Only a handful of attempts have been made to induce RODD structures directly from the data [128, 129]. It has also been proposed to use RODDs for knowledge presentation to humans [130].

2 Basic Definitions

This and the following sections are included for completeness of discourse, notation, and terminology. Many of the concepts that follow were introduced in [115, 131]. The order of presentation follows [121, 132].

Decision diagrams are data structures for manipulation of functions over finite domains. Assume that there is a set of attributes A in which each every attribute a takes a finite number of values belonging to a set V_a. There is a finite set of classes (or conclusions) C representing the range of the diagrams.

The terms "variable" and "attribute" will be used interchangeably because the ML community prefers to describe objects by their attributes, whereas the VLSI community follows the mathematical concept of functions depending on variables. A classifier is a function using attribute values of a given sample as values of variables

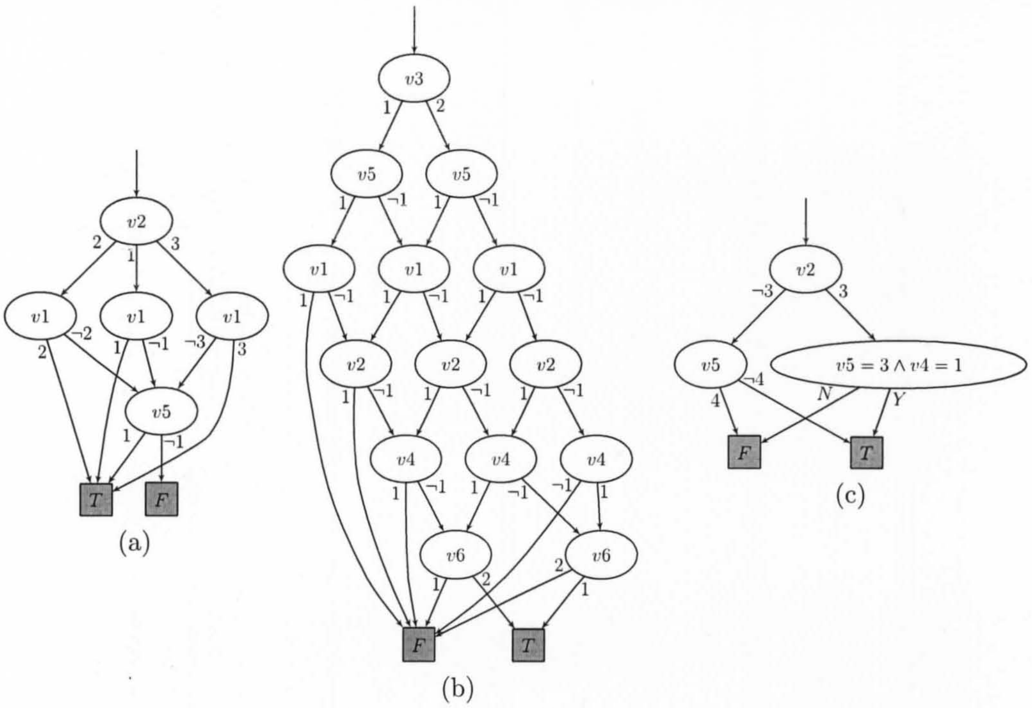


Figure 14: Decision diagrams for MONK's problems: (a) realizes $v1 = v2 \lor v5 = 1$, (b) realizes *exactly 2 of* $v1 \ldots v6$ *are* 1, (c) realizes $(v5 = 3 \land v4 = 1) \lor (v5 \neq 4 \land v2 \neq 3)$. Note that some nodes have been merged for better presentation, this affects only the display and does not change the internal representation of the diagrams [86]. © 2011 IEEE

in the function expressing it.

Conceptually, a decision diagram is similar to a decision tree: it consists of nodes in which tests for attribute values are made and of directed connections between them. An ordering is defined on attributes and every path in the diagram must traverse the nodes in exactly this order. This facilitates the detection of common subgraphs which can be merged. Example RODDs presenting solutions for the MONKS [133] problems are shown in Figure 14.

Definition 2. *A Decision Diagram (DD) is a multi-rooted, directed acyclic graph with:*

- *terminal nodes having out-degree 0 and belonging to a set of conclusions,*
- *attribute nodes having the property that each attribute node u has an associated attribute att(u) and its out-degree is equal to the number of different values att(u) may take.*

When all attributes and the class are binary the diagram will be called a Binary Decision Diagram (BDD). The $succ_0(u)$ *and* $succ_1(u)$ *are then called respectively the* low *and* high *child of u.*

A DD is Ordered (ODD) if on all the paths through the graph attributes respect a given linear order $a_1 < a_2 < \cdots < a_k$.

An ODD is Reduced if:

- *(uniqueness) no two distinct nodes u,v are associated with the same attribute a and for each value of a have the same successor, i.e.,*

$$att(u) = att(v) \text{ and } \forall_i succ_i(u) = succ_i(v) \implies u = v$$

- *(non-redundant tests) no attribute node u has all of its successors equal, i.e.,*

$$\forall_u \exists_{i,j} succ_i(u) \neq succ_j(u)$$

The RODDs are a canonical representation of functions, i.e. every function $f : I \to C$ has a unique (up to the variable ordering chosen) representation as a RODD.

3 Efficient Manipulation of ROBDDs

For simplicity, only the case of boolean functions is presented. However, most algorithms are easily transformed to support n-ary variables. For a boolean function $F : \mathbb{B}^n \to \mathbb{B}$ the cofactors of F denoted F_x and $F_{\bar{x}}$ are the results of applying to F the substitutions $[x\backslash 1]$ and $[x\backslash 0]$, respectively. By definition a function depends on variable x when $F_x \neq F_{\bar{x}}$. Assume that a total order on variables π is known. The topmost variable occurring in some expressions is the minimal, according to π, variable on which at least one of the given expressions depends.

Each node of a ROBDD serves as the entry point to the function represented by it. Function evaluation consists of following the path determined by attribute values until a terminal node is reached. If it is assumed that all functions are kept in the same graph, a shared ROBDD is created. Some form of garbage collection is needed to keep track of referenced nodes. Usually, reference counting is used, but more advanced implementations use mark and sweep and generational collectors.

According to the Shannon function expansion, a boolean function can be expressed as an if-then-else (ite) expression, i.e. for any $F : \mathbb{B}^n \to \mathbb{B}$

$$F = x \cdot F_x + \bar{x} \cdot F_{\bar{x}} = \text{ if } x \text{ then } F_x \text{ else } F_{\bar{x}} = ite(x, F_x, F_{\bar{x}})$$

In ROBDDs the children of a node are the cofactors of the function represented by it. Consequently, attribute nodes are often called Shannon nodes.

All boolean operations can be expressed in the form of a generalized if-then-else operator $ite(F, T, E) = F \cdot T + \bar{F} \cdot E$, taking three nodes of the shared ROBDD. The *ite* computation can be expressed in terms of the cofactors of F, T, and E with regard to their topmost variable. This leads to an elegant and efficient recursive implementation.

Two hash tables are used in algorithms manipulating ROBDDs. The first one, called the unique table, stores triples $(variable, highbranch, lowbranch)$ for all nodes. It is queried to ensure the uniqueness property. A new node is created unless an equivalent one is already present in the unique table. This mechanism results in $O(1)$ node retrieval (or creation if it didn't exist) time.

The second table caches results of executed operations: $(op\_code, operand, \dots)$. It guarantees that the *ite* operation running time is bounded by $O(|F| \cdot |T| \cdot |E|)$ where $|N|$ denotes the number of nodes reachable from a node N. It is just the total number of different calls to *ite* that may result in a particular computation.

4 Choosing and Changing the Variable Order

The efficiency of operations on ROBDDs depends on the number of nodes in the operands. This is closely tied to the chosen order on the variables. Some functions (notably the class of symmetric functions) require a similar number of nodes for any variable order. Many functions require polynomially many nodes (with regard to the number of attributes) under some orderings, and exponentially many under different orderings. Some functions (notably those representing middle bits of a multiplication) always require exponentially many nodes.

The practice of using ROBDDs in the VLSI domain shows that many commonly found functions can be efficiently processed when a good variable order is chosen. Determining the best ordering is NP-hard [134], but many good heuristic methods exist to approximate it. In the case of VLSI circuit design layout analysis techniques are used to provide an initial guess of the ordering. A heuristic designed for the case of learning from data is presented in the section B. Furthermore, the variable ordering used can be modified during program execution. The most popular methods are the sifting algorithm, which moves variables one-by-one through all possible positions, and the windowing algorithm which finds the best possible arrangement for small groups of adjacent variables (called windows) [135].

To enhance the reordering process, interactions between variables can be detected. A method for efficient grouping of neighboring variables is presented in [136]. A method for detecting when two variables do not interact and thus require no action to be swapped in the order is described in [137].

5 Extensions

Classical ROBDDs need exponentially many nodes to represent multiplier circuits. This deficiency triggered the development of many extensions which could reduce the size of diagrams for several classes of functions. Investigated were different meanings of nodes, e.g. the Shannon if-then-else nodes were replaced by moment decomposition, $f = f_{\bar{x}} + x \cdot (f_x - f_{\bar{x}})$ [119]. Other research focused on extending the range of the diagrams from single bits to words, integer or real values. In the simpler approach, each distinct value is assigned a terminal node leading to the "so-called" Algebraic DDs [138]. In the more elaborate approach, functions transforming return values are added to the edges. For example, complemented edges have a binary flag indicating that the result of the function represented by the node they point to is to be negated. A uniform treatment of diagrams containing edges annotated with such functions is presented in [118]. With additional constraints the diagrams with edge-functions remain a canonical function representation and can be used for efficient operation. The most popular approaches are: decision diagrams with complemented edges; EVDDs for integer functions [116]; Factored EVDDs [117];

identical, but introduced independently Affine Algebraic Decision Diagrams [139] and Normalized Algebraic Diagrams [118].

The addition of edge-values enlarges the class of functions which can be expressed using polynomially sized decision diagrams. However, this may come at the cost of an increase in the computational complexity of operations (even exponentially), because the number of invocations of any algorithm may depend not only on the number of combinations of nodes, but also on the number of different edge-value compositions obtained while reaching those nodes.

Attempts to relax the ordering restriction resulted in a concept of types which generalizes the linear ordering [140, 141]. This variant is called Free Decision Diagrams (FDDs). Decision diagrams are often used in research on Petri Nets, model checking and constrained problems. A potentially useful for rule processing extension describes operations on linear constrains of the form $x - y < const$ [142].

B Extracting Rules from Neural Networks as Decision Diagrams [1]

In this section a method that induces an RODD from a trained neural network is presented. The considerations of the computational complexity of the problem of rule extraction presented in Chapter II has shown that it is computationally infeasible to provide an exact replica of the network. To reduce the computational complexity, the method presented in this section limits the analysis to the parts of the data space close to the training set samples. The rule extraction problem has been restricted to *provide a description of the function realized by the neural network near the samples from the training set.* The rationale for this restriction was that first, the ultimate goal is to learn from the data, not from the network. Second, the network itself finds some relations in the training set. The further away the data are, the more complicated the decision surface of the network might be and the more unnecessary it becomes to faithfully describe it.

[1]This material has been presented in [86], (J. Chorowski and J. M. Zurada, "Extracting Rules from Neural Networks as Decision Diagrams," *Neural Networks, IEEE Transactions on*, vol. 22, no. 12, pp. 2435–46, Dec. 2011, © 2011 IEEE).

Algorithm 3: Outline of the method

```
1: $G \leftarrow \boldsymbol{U}$ {start with the empty rule}
2: for all $\boldsymbol{x} \in$ Training Set do
3:     {Create new partial rule describing just this sample}
4:     $PR \leftarrow derivePR(net, \boldsymbol{x})$
5:     $G \leftarrow merge(G, PR)$
6: end for
7: {now $G$ correctly classifies all training samples and needs to be
   extended over whole feature space}
8: $G \leftarrow generalize(G)$
9: $Pruned \leftarrow prune(G)$ {optionally further simplify}
```

1 Detailed Description of the Proposed Method

The outline of the algorithm is presented in Algorithm 3. It first captures network's actions for each training sample and expresses it in the form of a partial rule. It then proceeds to merge the partial rules into a single classifier using the RODD methodology. Finally, it applies generalization and pruning operations to reduce the resulting diagram size.

To simplify the description of the proposed method this discussion is limited to the case of two classes (denoted by $\mathcal{T}$ and $\mathcal{F}$) computed by a network having only discrete inputs and only one hidden layer. The algorithms are easily extended to handle multi-class problems, as well as more complicated network structures. However, no continuous attributes are supported.

The notation is as follows: let the network have k input *attributes* and let A_i denote the i-th attribute, taking n_i different values. When there is no confusion A_i will also be used to denote the set of values taken by the i-th attribute. The space of network inputs, or the attribute space, is thus $\mathcal{I} = A_1 \times A_2 \times \ldots \times A_k$. For every attribute A_i, the network has n_i *inputs* that use the 1-of-N encoding. All network inputs take either the value 0 or 1.[2] For an attribute A and a vector $\boldsymbol{x}$, let $\boldsymbol{x}_A$ denote the part of $\boldsymbol{x}$ associated to that attribute (the input values of encoded attributes A, the weights connecting such inputs, etc.). Let $K = \sum_{i=1}^{k} n_i$ be the total number of

[2]For training purposes more suitable values would be ± 1. The weights can be linearly scaled to accommodate a different input encoding after training.

inputs. The network realizes a numerical function $\bar{f} : \mathbb{R}^K \to \mathbb{R}$. However, the only *valid* (meaningful) inputs are binary vectors of length K in which for every attribute there is exactly one input having value 1 and all the others have value 0. Consequently, an input is by definition invalid when it is not valid (i.e. for at least one attribute all the inputs are zeroed, or more than one input is active). $\bar{\boldsymbol{x}}$ is to denote a real vector which is a valid network input corresponding to an input sample $\boldsymbol{x} \in \mathcal{I}$. For those valid vectors the logical function $f : \mathcal{I} \to \{F, \mathcal{T}\}$ is defined in the usual way as:

$$f(x) = \begin{cases} \mathcal{F} & \text{if } \bar{f}(\bar{\boldsymbol{x}}) < 0, \\ \mathcal{T} & \text{if } \bar{f}(\bar{\boldsymbol{x}}) \geq 0. \end{cases} \tag{54}$$

During the execution of the algorithm partial rules are deduced. They are functions assigning to samples either the class label, or the special value $\boldsymbol{U}$ (unknown) denoting that the sample isn't classified by the rule.

Definition 3. *A* partial rule *PR is a function $PR : \mathcal{I} \to \{\mathcal{F}, \mathcal{T}, \boldsymbol{U}\}$ from the attribute space into the set of classes augmented with the special value* unknown, *denoted as $\boldsymbol{U}$. Moreover the* domain *of a partial rule is defined to be the the subspace of valid inputs for which the partial rule's value is known:*

$$Dom(PR) = \{\boldsymbol{x} \in \mathcal{I} \mid PR(\boldsymbol{x}) \neq \boldsymbol{U}\}$$

Hence while a rule can be applied to all samples, only those belonging to that rule's domain will be classified.

Two rules will be said to agree (denoted by $\approx$) if they identically classify samples belonging to the intersection of their domains:

Definition 4. *Two partial rules $R1$ and $R2$* agree *with each other if and only if:*

$$\forall_{\boldsymbol{x} \in \mathcal{I}} \boldsymbol{x} \in Dom(R1) \wedge \boldsymbol{x} \in Dom(R2) \Rightarrow R1(\boldsymbol{x}) = R2(\boldsymbol{x})$$

Two partial rules that agree can be merged into a new one whose value will be known on the union of their domains and agreeing with both of them.

Definition 5. *A partial rule R is the result of* merging *two partial rules that agree, $R1$ and $R2$, if and only if:*

$$R1 \approx R2$$

$$Dom(R) = Dom(R1) \cup Dom(R2)$$

$$R \approx R1 \text{ and } R \approx R2$$

Obviously, the network function f as given in (54) is a partial rule whose domain covers the entire attribute space. The constant function $U(X) = \boldsymbol{U}$ has an empty domain and is not useful for classification, but it agrees with every other partial rule.

The algorithm presented in Algorithm 3 starts with a partial rule $G = \boldsymbol{U}$. Then in a loop over each training sample the domain of G is extended to cover the processed sample and a part of its neighborhood. It can be proved that G agrees at every step with the network function and that its domain contains all the processed training samples. This means that whenever the value of G at $\boldsymbol{x}$ is not $\boldsymbol{U}$, it is equal to the network function value, i.e. $G(\boldsymbol{x}) \neq \boldsymbol{U} \implies G(\boldsymbol{x}) = f(\boldsymbol{x})$. However, if $G(\boldsymbol{x}) = \boldsymbol{U}$, then the rule G doesn't provide any information about the class of $\boldsymbol{x}$.

Upon the completion of the loop over the training set, G holds a partial rule which classifies every training sample and generalizes over some part of the input space in a similar way to the network. Next, to extend G's domain to the full input space, a generalization procedure can be applied. Please note that this step breaks compatibility with the network and the generalization G' of G does no longer agree with f, the network function (the rule set and network may disagree on previously unseen samples). Generalization can be followed by an optional simplification (pruning) step, which furthermore may break the agreement with the network on the training set.

Estimating Minimal and Maximal Network Excitation.

A partial rule is derived from each training sample $\boldsymbol{s}$ by determining a subset of attributes that are sufficient to classify this sample. These attributes are called the *important* attributes of $\boldsymbol{s}$. They are found through estimation of network's output when the value of some attributes is not set and greedy removal of attributes whose elimination does not change the network's classification. The new partial rule used to classify a previously unseen sample $\boldsymbol{x}$ derived from a training sample $\boldsymbol{s}$ is then:

$$PR_{\boldsymbol{s}}(\boldsymbol{x}) = \begin{cases} Class(\boldsymbol{s}) & \text{if } \forall_{A \in Important(\boldsymbol{s})} \boldsymbol{s}_A = \boldsymbol{x}_A \\ \boldsymbol{U} & \text{otherwise.} \end{cases} \tag{55}$$

Meaning that if a sample $\boldsymbol{x}$ and a known training sample $\boldsymbol{s}$ have exactly the same value of all important attributes of $\boldsymbol{s}$, then $\boldsymbol{x}$ is classified in the same way as $\boldsymbol{s}$. Otherwise, the class of $\boldsymbol{x}$ is unknown to the rule and $\boldsymbol{U}$ is returned.

Before delving into the case of a whole network, a single neuron will be analyzed. Len N be a neuron having activation function:

$$N(\boldsymbol{x}) = \sigma(\boldsymbol{x} \cdot W - b) = \sigma(\sum_{A=1}^{k} \boldsymbol{x}_A \cdot W_A - b) \tag{56}$$

where $\sigma(.)$ is a sigmoidal transfer function, $\boldsymbol{x} \in \{0,1\}^K$ is a valid input vector, W is the weight vector and b is the bias, and $\boldsymbol{x}_A$ denotes the part of $\boldsymbol{x}$ describing the attribute A.

Let A be a selected attribute. Consider the neuron N' with weights W' and bias b' obtained from N by subtracting from weights associated with the attribute A and the bias the minimum weight for that attribute, i.e.

$$W'_i = \begin{cases} W_A - min(W_A) & \text{for } i = A \\ W_i & \text{for } i \neq A \end{cases} \tag{57a}$$

$$b' = b - min(W_A). \tag{57b}$$

It will be shown that for all valid inputs (those having the property that for every attribute exactly one of its associated inputs is 1, while the others are 0) the excitation of N' is equal to that of N, hence the two neurons are equivalent. However, N' can be used to calculate the minimum excitation of N if the value of a attribute A is not specified by calculating the neuron's excitation for an input vector $\boldsymbol{x}'$ with all inputs associated with the attribute A zeroed.

Theorem 3. *For any valid input $\boldsymbol{x}$, the neuron N' obtained from a neuron N using the transformation* (57) *has exactly the same activation value. Furthermore for an input vector $\boldsymbol{x}'$ obtained from $\boldsymbol{x}$ by zeroing inputs associated with the attribute A, i.e. $\boldsymbol{x}'_A = 0$, $\boldsymbol{x}'_i = \boldsymbol{x}_i$ for $i \neq A$ the neuron N' returns the minimum activation of N for all possible values of A.*

Proof. Without loss of generality the attributes can be reordered, so that the attribute A is the first attribute. If the input vector $\boldsymbol{x}$ is valid, then exactly one input associated with the attribute is set to 1, while the others are zeroed, i.e. $\boldsymbol{x}_1$ has exactly one 1 on the J-th position. Thus exactly one of W_1, weights associated with the first attribute, is included in the summation. Then

$$\begin{aligned} \boldsymbol{x} \cdot W - b &= \sum_{j=1}^{n_1} \boldsymbol{x}_{1,j} W_{1,j} - b + \sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i = W_{1,J} - b + \sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i = \\ &= (W_{1,J} - min(W_1)) - (b - min(W_1)) + \sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i = \\ &= \sum_{i=1}^{k} \boldsymbol{x}_i \cdot W'_i - b' = \boldsymbol{x}' \cdot W - b', \end{aligned} \tag{58}$$

since the only nonzero element in $\boldsymbol{x}_1$ is $\boldsymbol{x}_{1,J} = 1$.

To prove the second part observe that:

$$\begin{aligned} \min_{\boldsymbol{x}_1} N(\boldsymbol{x}) &= \min_{\boldsymbol{x}_1} \sigma \left(\boldsymbol{x}_1 \cdot W_1 + \sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i - b \right) = \\ &= \sigma \left(\min(W_1) + \sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i - b \right) = \\ &= \sigma \left(\sum_{i=2}^{k} \boldsymbol{x}_i \cdot W_i - (b - \min(W_1)) \right) = N'(\boldsymbol{x}') \end{aligned} \tag{59}$$

□

By repeatedly applying the transformation (57) to all attributes, a neuron returning equal excitation values for all valid inputs and the minimum excitation for an input vector with some attributes zeroed is obtained. To estimate the maximum excitation if some attributes are omitted the maximum is subtracted instead of the minimum in equation (57). The resulting transformation of a neuron N with weights W and bias b into neurons N_{min} and N_{max} estimating the minimum and maximum excitations, respectively is:

$$\begin{aligned} W'_{min} &= \forall_{A\in 1\ldots k}\ W'_{minA} = W_A - \min(W_A) \\ b'_{min} &= b - \sum_{A=1}^{k} \min(W_A). \end{aligned} \tag{60a}$$

$$\begin{aligned} W'_{max} &= \forall_{A\in 1\ldots k}\ W'_{maxA} = W_A - \max(W_A) \\ b'_{max} &= b - \sum_{A=1}^{k} \max(W_A). \end{aligned} \tag{60b}$$

Using the neuron transformations (60) it becomes easy to test if the neuron's output is independent of some attributes. It suffices to zero the inputs associated with the supposedly unimportant attributes, and check that the maximum and minimum neuron excitations have the same sign.

To calculate the minimum excitation of a multilayer network having just one hidden layer the transformation (60) is first applied to all neurons in the hidden layer. The computation is then performed for the output neuron ON by calculating:

- the minimum excitation of neurons that are connected to ON with positive weights,
- the maximum excitation of neurons that are connected to ON with negative weights.

In the case of a more complicated network architecture the above procedure can be repeated recursively. Please note that the transformation (60) gives an exact value for the minimum/maximum excitation of a single neuron. For a multilayer network it gives an upper bound for the minimum excitation and a lower bound for the maximum.

Deriving Partial Rules from Input Samples

The procedure shown in Algorithm 4 is used to obtain a partial rule classifying a given sample. The main part of the algorithm searches for a small set of important attributes. First, an ordering of attributes has to be selected. Best results

Algorithm 4: Extract a partial rule for a given sample.

```
1: fun $derivePR(net, \boldsymbol{x})$
2: $AttributeList \leftarrow$ Ordered attributes
3: $Important \leftarrow Empty$
4: $\boldsymbol{x}' \leftarrow \boldsymbol{x}$
5: for all $A \in AttributeList$ do
6:    $\boldsymbol{x}'_A \leftarrow 0$ {zero the inputs associated with attribute}
7:    if $\max(net, \boldsymbol{x}') \cdot \min(net, \boldsymbol{x}') < 0$ then
8:       {Network's output is unstable}
9:       add $A$ to $Important$
10:      $\boldsymbol{x}'_A = \boldsymbol{x}_A$ {restore the inputs for attribute $A$}
11:   end if
12: end for
13: if Exists $A \in Important$ such that class doesn't change for all values
    of $A$
     then
14:   remove $A$ from $Important$
15: end if
16: $PR \leftarrow$ new partial rule given by equation (55)
17: return $PR$
```

have been obtained when the attributes were considered for removal starting with the bottom most attributes of the decision diagram ordering. Then, according to that ordering, the algorithm tries to greedily remove attributes from the rule's antecedents. It then tries to eliminate one last attribute by checking if the network assigns the same class for all its possible values.

Choosing an Attribute Ordering

Choosing a good attribute ordering is very important. First, the better the ordering, the smaller the resulting diagram will be. Second, the extracted diagram might generalize better, as partial rule derivation, pruning, and generalization procedures work bottom-up and rarely change connections near the top of the diagram. A heuristic procedure can be developed using two assumptions. First, attributes which often belong to the same rules should be close in the ordering. Second, the most important attributes should be placed near the top of the diagram.

To determine the saliency of attributes, the following measures were analyzed:

Algorithm 5: The default attribute saliency estimation.

```
fun $saliency(net, \boldsymbol{x})$
$Ret[] \leftarrow zeros(\#$ of attributes$)$
for all $A \in Attributes$ do
  for all $v \in values(A)$ do
    if $v = \boldsymbol{x}_A$ then
      *skip*
    end if
    $\boldsymbol{x}' \leftarrow \boldsymbol{x}; \boldsymbol{x}'_A \leftarrow v$
    $Ret[A] \leftarrow |net(\boldsymbol{x}) - net(\boldsymbol{x}')|$
  end for
  $Ret[A] \leftarrow Ret[A]/(\#$ of values of $A - 1)$
end for
return $Ret[]$
```

Algorithm 6: Heuristic to determine the variable ordering.

```
fun $attributeOrdering()$
$Ordering \leftarrow \emptyset$
local fun $processCluster(c)$
  if $c$ has only one element then
    append $c$ to $Ordering$
  else
    $cs \leftarrow$ sub-cluster of $c$ having $Strongest(c)$
    $co \leftarrow$ the other sub-cluster of $c$
    $processCluster(cs)$
    $processCluster(co)$
  end if
end fun
$SumDist[] \leftarrow 0$
$SumSaliency[] \leftarrow 0$
for all $\boldsymbol{x} \in$ Training Set do
  $Saliencies[] \leftarrow saliency(net, \boldsymbol{x})$
  $SumSaliency[] \leftarrow SumSaliency + Saliencies$
  $SumDist[] \leftarrow SumDist + distances(Saliencies)$
end for
$Clusters \leftarrow hierarchicalCluster(SumDist)$
for all $c \in Clusters$ do
  $Strongest[c] \leftarrow$ most salient attribute in $c$
end for
$processCluster($top-level cluster from $Clusters)$
return $Ordering$
```

TABLE 6

Truth table used for the merge operation. © 2011 IEEE

A	B	$merge(A, B)$
$\boldsymbol{U}$	X	X
X	$\boldsymbol{U}$	X
X	X	X
X	Y	$X \neq Y \implies$ error

the perturbation method [87], and the maximum, minimum and average difference between the neuron's output for all of an attribute possible values. The best option proved to be the average difference of network output for all values of a attribute (as shown in Algorithm 5) and that has been selected as default.

The developed heuristic performs three main steps. First, for every sample belonging to the training set attribute saliencies are determined. Distances between these are calculated and added together. Next, a hierarchical clustering algorithm is run on the cumulative distances to find which attributes are often present together. In each cluster, the most salient attribute is determined. Finally, the ordering can be derived. The most salient attribute is chosen first. Then, if there are more attributes in its cluster, the most salient one is selected. Otherwise, the most salient attribute from the next cluster is taken. Details are shown in Algorithm 6.

Merge Operation

The merge operation has been implemented as a binary operation following the *apply*(.) function pseudo code taken from [132]. The truth table used is shown in Table 6. Note that the merge operation should only be run on agreeing diagrams, otherwise the result will be undefined (the merge of disagreeing partial rules results in an error).

Generalization

If enough training set samples are available and all partial rules have been merged the resulting diagram should cover the whole input space. However, often

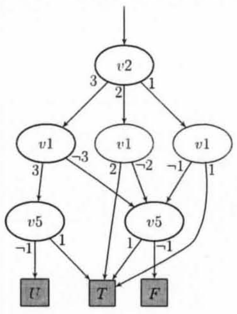


Figure 15: An incomplete decision diagram for the first MONK's problem after processing a half of the training set. © 2011 IEEE

Algorithm 7: Diagram generalization by rerouting paths leading to the $\boldsymbol{U}$ node

```
1: fun $generalize(Node)$
2: $Chld[] \leftarrow$ children of $Node$
3: $FC \leftarrow$ most frequently followed child $\neq \boldsymbol{U}$
4: for all $c \in Chld$ do
5:    if $Chld[c]$ points to $\boldsymbol{U}$ then
6:       $Chld[c] \leftarrow Chld[FC]$
7:    end if
8: end for
9: for all $c \in Chld$ do {recursively generalize children}
10:    $Chld[c] \leftarrow generalize(Chld[c])$
11: end for
12: return $Mk(Var(Node), Chld)$
```

several nodes point to the $\boldsymbol{U}$ terminal node (compare Figure 14a with Figure 15), indicating that for some inputs we don't know the answer. There can be several strategies to solve this problem. A solution is to perform some form of heuristic redirecting of paths leading to the $\boldsymbol{U}$ node. A suitable algorithm based on the $simplify(.,.)$ procedure from [132] is presented in Algorithm 7. It assumes that path usage counts have been recorded on the training set prior to its execution. It then proceeds in a top-down manner starting from the root of the diagram. Whenever a child pointer leads to the $\boldsymbol{U}$ node, it is redirected to its most frequently used sibling. The $mk(.)$ function, described in detail in [132] creates a new node ensuring that the diagram is reduced, i.e. it detects isomorphic subtrees and eliminates nodes whose all children are identical. For example, in Figure 15, the path
$v2 = 1 \land v1 = 1 \land v5 \neq 1 \rightarrow \boldsymbol{U}$ would be changed to

TABLE 7

Efficiency of the attribute ordering heuristic. Averaged results for 100 runs. © 2011 IEEE

Min size	Max size	Avg. size	Standard deviation	Heuristic size
8.47	14.58	10.89	1.45	9.28

$v2 = 1 \wedge v1 = 1 \wedge v5 = any \rightarrow \mathcal{T}$, which in turn would be simplified by $mk(.)$ to $v2 = 1 \wedge v1 = 1 \rightarrow \mathcal{T}$. In this case the heuristic would make the right choice.

Pruning

In many cases, the generated decision diagrams classify all samples, but they are overly complex. The pruning procedure is used to reduce the size of the diagram, while preserving its output on all training samples, or a majority of them. The main idea is simple: first, count how often a path was selected for all the elements in training set. Second, change the seldom (according to a selected threshold) or never used edges to point to the $\boldsymbol{U}$ node. Third, run the generalization procedure. In the current implementation, pruning is repeated as long as it reduces the size of the diagram.

Various Enhancements

Several simple heuristics have been applied during the development of the software to improve its performance. The most important one controls how much information is extracted from a single sample. By setting an option partial rules are also derived for all neighbors (in the Hamming distance sense) of the currently considered sample.

2 Experimental Results

The method was first tested on 100 randomly generated, simple logical formulas. Each formula had six attributes and was expressed in a DNF form having 8 clauses, each containing on the average 3.5 literals. For each formula, the best

TABLE 8

Results on the MONK's tests. © 2011 IEEE

Test	Network			LORE rules unpruned			LORE rules pruned		
ID	errors	time [s]	HL size	errors	size	time [s]	errors	size	time [s]
1	0	3	4	0	7	0.4	0	7	0.6
2	0	3.2	4	7.1%	39	0.44	18.5%	19	0.93
2^3	0	3.2	4	0	16	0.45	0	16	0.67
3	4.6%	1.4	1	4.6%	9	0.5	2.8%	4	0.9
3^4	2.8%	0.74	1	2.8%	4	0.38	2.8%	4	0.6

ordering had been found by testing all the possibilities and the smallest diagram size was compared to the one obtained with the heuristic for attribute ordering. Results have been shown in Table 7. On the average, the attribute ordering heuristic produced graphs having 9.28 nodes, while the average size of the diagrams for a random ordering is 10.58 with a standard deviation 1.45. Hence the attribute ordering selected by the heuristics produced on the average diagrams one standard deviation smaller than those using a random ordering. For 58 out of 100 cases, the heuristic procedure resulted in diagrams having the minimum size. Furthermore, it never led to a diagram having the maximum possible size.

As a second test, the ubiquitous MONK's [133] problems have been used. The three tests use the same data, consisting of six categorical attributes, $v1 \ldots v6$, and only the relation used to classify samples is changed. In the first test it is $v1 = v2 \vee v5 = 1$, in the second it is *exactly* 2 *attributes are* 1, and in the third $(v5 = 3 \wedge v4 = 1) \vee (v5 \neq 4 \wedge v2 \neq 3)$. Moreover, in the third test five percent of training samples have incorrect labels. In all the tests neural networks were trained using weight decay. Also, the diagram was pruned by removing paths used less than three times. The results are shown in the Table 8. Good decision diagrams are shown in Figure 14. While there is enough training set samples to generate a diagram that covers the whole input space in the first and third tests, in the second test the diagram is incomplete – even though the network has learned without errors, the decision diagram has a high error-rate. In this case, adding information about the neighborhood of a sample solves the issue. Compare Figure 14b showing a good

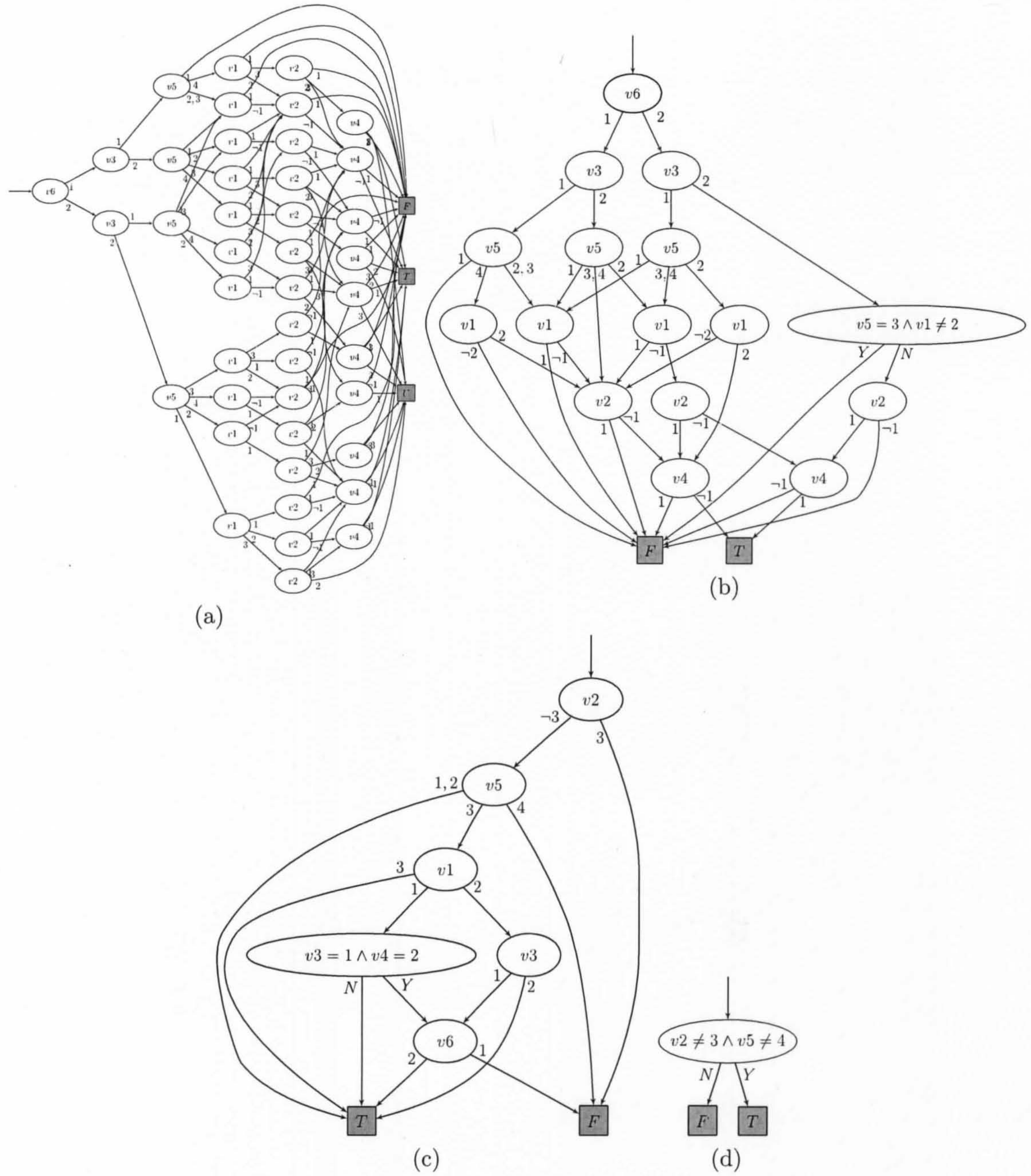


Figure 16: Decision diagrams for the Monk's [133] problems: (a) test 2 before diagram generalization, (b) erroneously pruned diagram for test 2 (proper diagram is shown in Figure 14b), (c) test 3 prior to pruning, and (d) test 3 after pruning (proper diagram is shown in Figure 14c). © 2011 IEEE

diagram with Figure 16a showing a diagram before generalization and Figure 16b depicting the results of pruning. In the third test, the training set is intentionally corrupted with noise. Note that prior to pruning, the diagram exactly mimics the network (the error rate is the same). The learned relation is more complicated than the expression used to generate the data. Retraining the network with a more aggressive weight decay or pruning the diagram both lead to a diagram which generalizes better, but only one clause out of two is properly detected. Compare Figure 16c showing an unpruned diagram with the pruned one in Figure 16d. Please note that the network and subsequently the diagram learns only half of the desired relation. The good diagram is shown in Figure 14c.

The next four tests used data from the UCI repository [17]: the mushroom data set, the congressional voting records data set, the chess king-rook vs. king-pawn data set (further named krkpa7) and the molecular biology promoter gene sequence data set. Since the method doesn't support missing values, samples from the voting records data set containing missing values were rejected. In the mushroom set, the missing values were left as a new attribute value "?". For all the sets, unless noted otherwise, five runs of full five-fold cross validation were executed and the results have been averaged. Also, pruning has been set up to preserve 100% accuracy on the training set. To compare the effectiveness of rule extraction, the C4.5 algorithm was run using Weka's J48 implementation [15]. Its parameters were *-U -M1* for runs without pruning and *-C0.25 -M2* for runs with pruning. The results have been presented in Table 9[5]. Diagram and tree sizes were used to compare the understandability of extracted rules.

The mushroom and voting sets were used as simple benchmark tests. On the mushroom data all classifiers obtain a 100% accuracy. The experiments on the voting data show the importance of good network training. When 10 hidden neurons are used the network overfits and yields a lower accuracy than the decision trees. In

[5]The results presented in Table 9 differ from those presented in [86]. They were recomputed because the original results mistakenly reported the sum of training and testing errors. Moreover, error counts were converted to accuracy for easier comparison with other methods.

TABLE 9: Results on data from the UCI repository.

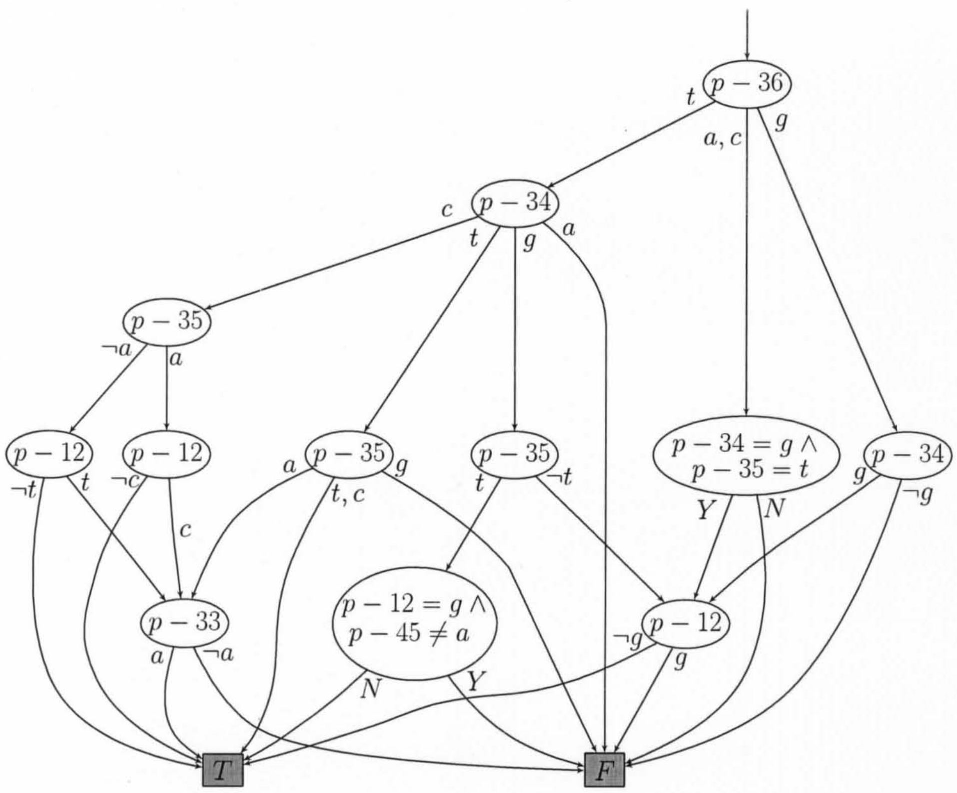
Test	Network			LORE rules unpruned			LORE rules pruned			J48 unpruned		J48 pruned		avg.
name	acc [%]	HL size	time [s]	acc [%]	size	time [s]	acc [%]	size	time [s]	acc [%]	size	acc [%]	size	time [s]
Mushrooms[a]	100.0	10	10.8	100.0	421.0	81.4	100.0	11.9	81.9	100.0	28.5	100.0	28.5	0.17
Voting[a]	94.0	10	0.8	93.5	75.0	0.5	94.7	20.2	1.0	94.2	23.6	96.6	4.5	0.01
Voting[a]	95.9	1	0.5	96.3	37.0	0.4	96.8	7.0	0.9	95.3	24.2	96.4	4.5	0.01
Krkpa7[b]	99.5	35	82.7	96.1	3838.9	18.0	95.9	494.8	18.5	99.5	88.6	99.3	51.7	0.09
Promoter[c]	90.6	1	0.8	89.4	332.3	2.2	81.1	15.0	2.8	74.3	35.9	77.2	20.4	0.01
Promoter[cd]	90.8	1	1.0	87.5	808.8	2.5	85.1	172.6	3.3	73.4	37.8	77.4	20.8	0.01
Promoter[ce]	94.3	1	2.6	97.5	405.7	2.7	90.6	209.1	3.3	77.4	44.5	81.1	24.7	0.00
Promoter[cde]	94.3	1	1.0	98.1	389.3	2.7	86.8	16.8	3.3	77.4	44.5	81.1	24.7	0.00

[a]Network trained using standard backpropagation.
[b]Network trained with weight decay, ratio=0.9
[c]Network trained with weight decay, ratio=0.3
[d]Merging in rules from training set samples neighborhood
[e]Tested using the leave-one-out methodology

consequence the extracted diagrams are also not as accurate as the trees. Smaller networks (with only one hidden neuron) have better cross-validation accuracy and the diagrams have accuracies and sizes comparable to the trees. Pruning is necessary on both datasets and it significantly reduces the size of the diagrams while preserving or improving their accuracy. It is worth noting that in both cases rule extraction running times are comparable to that of network training. The decision tree, however, is induced an order of magnitude faster than network training and rule extraction.

The results on the krkpa7 set are disappointing. The generated diagrams are not only big and hard to understand, but also their performance is worse than that of the network or of the decision trees. However, the issues may stem from some incompatibility between this data set and neural networks. The data set requires a network of a larger size (networks with few hidden neurons do not learn the relation well), with long training times (mainly due to the weight decay mechanism used), only to achieve a performance comparable to that of an unpruned decision tree. The results from this data set are included, however, to show that the method execution time is acceptable even for moderately large networks.

The promoter data set was first introduced to test the effectiveness of the KBANN method. Results published in [143] for a leave-one-test state that the ID3 method makes 19 errors in 106 runs, which corresponds to 82% accuracy, neural networks under standard backpropagation makes 8 errors (92.4% accuracy) and the KBANN method achieves the lowest rate of 4 errors (96.2% accuracy). For the purpose of this comparison, the LORE method was tested under the same conditions. However, since a single neuron with weight decay performs better than the described network, it was chosen as the base for rule extraction. It can be observed that the proposed method under the default settings extracts diagrams that have slightly worse accuracy than the network (prior to pruning). However, when the information from neighboring samples is included, the diagrams before pruning show the highest accuracy of 98.1%. The unpruned diagrams are too big to be considered understandable. Pruned diagrams can be considered to be legible, they are however less accurate. The experiments demonstrate that the idea of using a network to



Extracted rules for class $\mathcal{T}$:

$$
\begin{aligned}
&p-36=t \wedge p-35=t,c \wedge p-34=t\\
&p-36=g \wedge p-34=g \wedge p-12\neq g\\
&p-36=t \wedge p-35=a \wedge p-34=t \wedge p-33=a\\
&p-36=t \wedge p-35=t \wedge p-34=g \wedge p-12\neq g\\
&p-45=a \wedge p-36=t \wedge p-35=t \wedge p-34=g\\
&p-36=t \wedge p-35\neq t \wedge p-34=g \wedge p-12\neq g\\
&p-36=t \wedge p-35\neq a \wedge p-34=c \wedge p-12\neq t\\
&p-36=t \wedge p-35\neq a \wedge p-34=c \wedge p-33=a\\
&p-36=t \wedge p-35=a \wedge p-34=c \wedge p-12\neq c\\
&p-36=t \wedge p-35=a \wedge p-34=c \wedge p-33=a\\
&p-36=a,c \wedge p-35=t \wedge p-34=g \wedge p-12\neq g
\end{aligned}
$$

Figure 17: A typical diagram and the rules in DNF form extracted for the promoter domain problem without the use of neighboring samples and with pruning enabled. © 2011 IEEE

discover rules in close proximity to training set samples is useful, however better pruning and diagram generalization algorithms are needed. A sample diagram and accompanying rules extracted during the leave-one-test without the use of neighboring samples and after pruning is shown in Figure 17.

3 Conclusions

The LORE method of rule extraction from neural networks uses many novel ideas. First, the proposed approach focuses on retaining high network fidelity on the training set, while allowing the rule set to diverge from the network in the remaining feature space, making it possible to reconcile the dilemma whether one seeks good network fidelity or accuracy.

Another achievement is the adaptation of the reduced, ordered decision diagram data structure to support the merge and generalization algorithms. This, in its own merit, might prove to be a valuable tool in other rule induction schemes.

A mathematically sound method of presenting the domain of a generated rule set has been introduced. Adaptations of this technique to other rule extraction methods might help to distinguish between errors in the rules (which result in false positive classifications) and incompleteness of the rules (which result in false negative classifications). In the case of the LORE method, this technique was a key step in the design of algorithms that simplify the rule set without loss of training accuracy.

Future work will first be directed towards the development of better pruning algorithms. The minimum description length principle (already investigated in the context of decision diagrams in [129]) might become a valuable tool for increasing the generalization abilities of decision diagrams, while at the same time reducing their size.

Improved rule presentation is another important topic worth continued studying. The presented transformation of RODDs into DNF format rules shows that the RODDs might be used as intermediate representation purely for their algorithmic properties and the final rules presented to the user may be expressed in a more understandable form of decision trees, decision tables or rules.

C Top Down Induction of RODDs[6]

The rule extraction method described in the previous section searched for rules in a general-to-specific manner, because each training sample seeded a rule from which tests were dropped. Moreover the best results were obtained when the tests were considered for dropping starting with attributes situated at the bottom of the chosen ordering. This procedure can be replaced by a general-to-specific building of the diagram in a top-down manner. Similarly to decision tree induction, the search space can be limited by the size of the dataset - empty nodes are marked with the $\boldsymbol{U}$ label and are not further divided.

Under another interpretation, just as the TREPAN [73] method builds a decision tree using the given classifier for guidance, it is possible to build a RODD using a described elsewhere procedure [145], but also using the black-box classifier for stopping criteria and detection of similar regions in the search space.

An important attribute of the method described in this chapter is the detection of similar regions of the attribute space and the merging of corresponding RODD nodes prior to their expansion.

1 Description of the Algorithm

To reduce noise in the training set, prior to algorithm execution training samples may be reclassified. The algorithm then induces a decision diagram in a top-down manner, as shown in Algorithm 8. In every iteration of the loop a layer of the diagram is built. First, the splitting attribute is determined based on information gain with gain ratio correction. An attribute is selected once for each layer in the diagram as described in [13, 145]. Second, a new layer is created according to the split found. However, pure nodes are not splits. Third, nodes are merged based on similarity of input neurons activations. The algorithm then proceeds to build the next layer.

[6]This section is based on [144], (J. Chorowski and J. Zurada, "Top-down induction of reduced ordered decision diagrams from neural networks," *Lecture Notes in Computer Science, Artificial Neural Networks and Machine Learning- ICANN 2011*, vol. 6729, pp. 309–316, 2011).

Algorithm 8: Top down induction of RODDs

```
Require: A dataset $data$ with $k$ attributes and $N$ instances
Ensure: A decision diagram classifying $data$
 1: $currentLayer \leftarrow \{rootNode(data)\}$
 2: while some attributes remain untested and there are impure nodes
    do
 3:   $splitAttribute \leftarrow$ best unused attribute
 4:   $newLayer \leftarrow \{\}$
 5:   for all $node \in currentLayer$ do
 6:     $newLayer \leftarrow newlayer \cup \{split(node, splitAttribute)\}$
 7:   end for
 8:   for all $node \in newlayer$ do
 9:     check termination conditions for $node$, remove $node$ if met
10:   end for
11:   for all $n1, n2 \in newLayer, n1 \neq n2$ do
12:     compute distance($n1$,$n2$)
13:   end for
14:   while $n1, n2 = \underset{n1,n2}{\text{argmin}}\ distance(n1, n2)$ and $distance(n1, n2) <$
      $maxdistance$ do
15:     merge $n1$ with $n2$, recompute distances
16:   end while
17:   $currentLayer \leftarrow newLayer$
18: end while
```

Two stopping criteria are used. First, 100% pure node are not split. Nodes which do contain few (according to a set threshold[7]) or no training instances (which makes them trivially pure) are marked with an unknown class flag. Second, splitting is stopped when the network output is determined, i.e. upper and lower bounds of the network output computed using the values for already tested attributes have the same sign. The computation of the estimates follows Section B.1.1.

Two nodes are treated as suitable for merging if the root mean square of the difference of input neurons' partial activation is below a set threshold. The justification of node clustering is as follows. Let the contribution of a fragment of a path through the decision diagram towards neuron activation be

$$pathActivation(path) = \sum_{A \in \text{attributes tested on path}} x_A \cdot W_A, \tag{61}$$

[7]All experiments used the value 2.

where W_A are the weights used for attribute A. A partial activation of a neuron for a node is then

$$partActivation(node) = pathActivation(diagram\ root \rightarrow node). \tag{62}$$

Recall, that if values of all attributes are known a neuron's activation is

$$\begin{aligned} activation(leaf\ node) = \sum_{A \in \text{attributes}} x_A \cdot Weight_A + bias = \\ \forall_{node} pathActivation(diagram\ root \rightarrow node) \\ + pathActivation(node \rightarrow leaf\ node) + bias. \end{aligned} \tag{63}$$

Grace to the ordering restriction, all nodes in a single layer have known values for the same set of attributes. Hence if two nodes $n1$ and $n2$ have a similar partial activation the subdiagrams $s1$, $s2$ rooted at $n1$ and $n2$, respectively should be isomorphic. Thus $n1$ can be merged with $n2$. The new node contains the union of training samples reaching both $n1$ and $n2$, which helps to reduce fragmentation of the training set. After a merge operation the partial activations of all the nodes in a cluster are averaged.

During diagram construction, some leaf nodes may be marked with an unknown class label. Similarly to the method described in the previous chapter same diagram generalization and pruning procedures can be used.

To estimate the required number of operations observe first that the maximum width of the diagram (the number of nodes at a single level) is limited by the number of training samples because empty nodes are not split. It is thus bounded by $O(N \cdot l)$, where N is the number of training samples and l is the maximal number of values a attribute may take. Assume that l is small, then the width of the diagram becomes $O(N)$. For every diagram level node clustering takes $O(width^2)$ time and selection of the next attribute to split takes $O(k \cdot N)$ time, where k is the number of attributes. The total complexity is thus $O(k(k \cdot N + N^2))$ or $O(k \cdot N^2)$ if $k \ll N$.

2 Experimental Results

The proposed method has been tested on datasets from the UCI repository [17]. Used were the MONKS tests [133], the Voting, the German Credit,

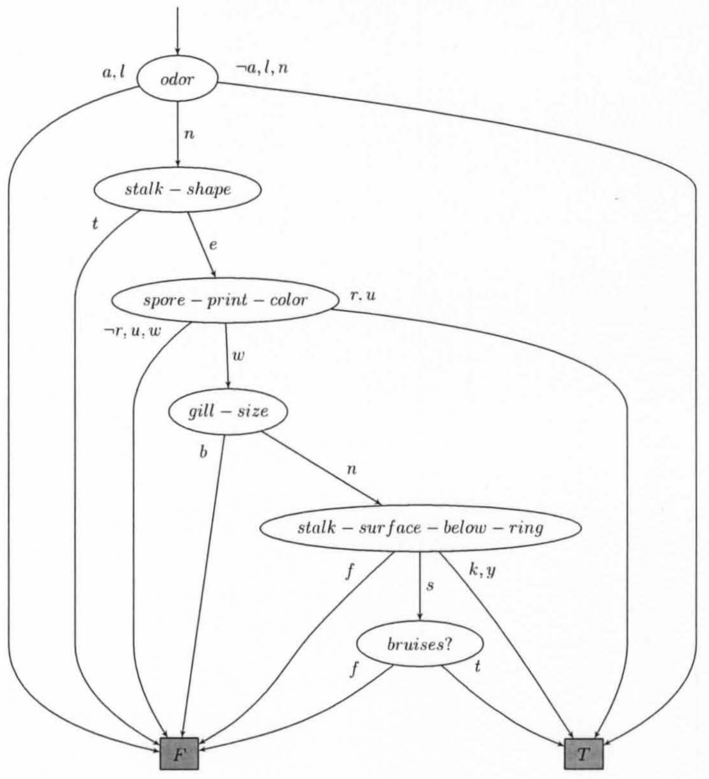


Figure 18: Induced decision diagram for the mushroom dataset.

the Mushroom dataset, and the Letter dataset. The proposed method currently supports only problems having nominal attributes. Therefore continuous attributes have been discretized using the Weka's [15] implementation of [146] with default parameters. Furthermore attributes taking more than 2 values were presented to the network using a 1-of-N encoding. Since binary classes are currently assumed, in the letter dataset the two classes were created by merging letters 1-13 (A-M) and 14-26 (N-Z). Low variance attributes were removed. Missing values are currently not supported. Consequently, the attribute 11 from the Mushroom dataset has been removed. Samples containing unknown attributes were removed form the Voting data. Moreover, to make the Voting problem harder highly informative attributes 1, 2, 4, 13, 16 were also removed (after [66]). Properties of the used datasets (after processing) are given in Table 10.

An exemplary RODD for the Mushroom dataset is shown in Figure 18. Results are presented in Table 11. The first columns show the number of neurons in the input layer and weight penalty ratio used during neural network training using

TABLE 10

Properties of the used datasets.

Dataset	train size	test size	attributes nom. & cont.	values total number
Monk1	124	432	6	15
Monk2	169	432	6	15
Monk3	169	432	6	15
Voting	324	10-CV	11	11
German	1000	10-CV	15	56
Mushroom	8124	10-CV	20	115
Letter	20000	10-CV	15	165

the Neural Network toolbox in Matlab. These parameters were tuned for a low cross-validation error. Next, results obtained using Weka's implementation of C4.5 (nicknamed J48) are presented along with accuracies for RODD reported in [145]. The rule extraction algorithm was then run with three different values for the node merging threshold – 0.2, 0.4, and 0.6. The best distance threshold found is indicated along with the resulting RODD accuracy, size, and the count of node merges performed. Since the used implementation of top-down RODD induction is currently much simpler than the one described in [145], for comparison its accuracy is reported also when NN information is absent.

The results show that especially for simple networks, node merging increases the accuracy of induced RODDs. However, for larger networks the gain in accuracy is small. Moreover, more aggressive node merging has been found to quickly worsen the accuracy. The diagrams are usually smaller than decision trees, however, their internal structure makes them harder to follow.

3 Possible Enhancements to Research

The presented approach is limited to the case of rule extraction form neural networks. Investigating other methods of detecting similar regions in the input space might enable its use to extract rules form other kinds of opaque classifiers.

TABLE 11: Results of experiments

	Network (NN)			J48		RODD	RODD from NN				RODD
Dataset	HL	wght.	acc.	acc.	size	acc.	dist.	acc.	size	merge	without
	size	pen.				[145]	thr.			count	NN acc.
Monk1	5	0.8	1.0	0.76	18	1.0	0.4	1.0	7	2	0.81
Monk2	5	0.8	1.0	0.65	31	0.68	0.4	0.98	16	14	0.65
Monk3	2	0.3	0.97	0.97	12	0.97	0.4	0.97	4	1	0.97
Voting	1	0.8	0.93	0.89	13.6	-	0.4	0.9	18.6	6.5	0.87
German	1	0.2	0.77	0.72	82.5	0.71[a]	0.2	0.76	51.5	73.1	0.72
Mushroom	5	0.8	1.0	1.0	30	1.0	0.2	1.0	8	0	1.0
Letter	40	0.6	0.94	0.87	3660	-	0.4	0.88	956	628	0.87

[a]results reported on non-discretized data

CHAPTER V

BUILDING DESCRIPTIONS OF DATA – FEATURE DETECTION

This chapter presents two methods related to feature detection and data encoding. In the first section the impact of non-negative weight constraints on the understandability of a neural network is considered. Then, a novel algorithm for finding robust sparse encodings of data is presented.

A Learning Neural Networks with Non-negative Weight Constraints[1]

Feature detection methods described in Chapter II found a transformation of the attributes into features that preserved the information content of the data. In the case of PCA the projection had to preserve as much as possible of variance of the samples, ICA concentrated on finding projections that retain interesting (non-gaussian) projections, and K-means, SVD, NMF, and sparse coding found constrained features that must provide an accurate reconstruction of the data. Thus all those methods work under an important assumption that useful features will describe the data and a direct relationship to class labels is not needed. It is a valid assumption when the original attributes used to describe the data are well chosen for the problem at hand. However, one may wish to develop other, new and special features that have good discriminative properties. The method presented in this section tries to accomplish just that.

Several authors have extended unsupervised feature detection algorithms with

[1]This section is based on [147] (J. Chorowski and J. M. Zurada, "Learning understandable neural networks with non-negative weight constraints," *Submitted to Neural Networks and Learning Systems, IEEE Transactions on*, 2012).

discriminative terms. Columns from a predetermined dictionary are selected to maximize both the Fisher discrimination coefficient and reconstruction capability in [148]. Per-class dictionaries are learned in [149], with a soft-max like penalty used to restrict the dictionaries to provide good reconstructions for the related class only. New samples are classified by choosing the label associated with the dictionary that yields the lowest reconstruction error. This idea is extended in [150] where a common dictionary is learned for all classes, however, additional penalty terms are added to promote discriminative capabilities. Since no explicit encoder is specified, encoding is performed simultaneously with classification by selecting the class that results in the lowest joint reconstruction and classification penalty.

Auto-encoding neural networks can be seen as feature detectors that are trained to minimize the reconstruction error and provide explicit formulas for both an encoder (that computes hidden neuron activations from the inputs) and a decoder (that computes the output from the hidden activations) [151]. Note that such definition of the encoder and decoder disagrees with the matching encoder or decoder specified using equations (24) or (25). Adding a traditional classification layer yields a semi-supervised neural network. A hierarchical model of text documents that greedily learns a stack of encoders is analyzed in [152]. The encoders assume that the data have a Poisson distribution and are a special case of auto-encoding neural networks. They are trained to minimize the sum of a reconstruction and classification error.

The architecture presented in this Section differs in two ways from the aforementioned extensions of feature detection. It is an extension of a traditional neural network and is trained for discrimination only. Moreover, the emphasis is placed on the understandability of the network's structure through the adoption of non-negative constraints on weights of the network.

Multilayer feed-forward neural networks naturally build hierarchical models of data [9]. The data samples are propagated through network layers and the neuron activations in each layer can be thought to form a layer of features. However, such features are usually difficult to interpret [23, 43] (also see the discussion of hierarchical models in Chapter II, Section F.2). On the other hand the non-negativity constraints have been shown to enhance the understandability of matrix factorizations [57]. Inspired by NMF, the network's weights are constrained to be non-negative. This eliminates cancellations of incoming neuron signals inside the network and allows for easier interpretation. Hidden (input) neurons are active when their inputs correlate strongly with their weights. The bias controls the threshold of this correlation. Classification (output) layer neurons combine the hidden layer activation values in direct proportion to their weights, and the neuron with the highest sum determines the class of the input.

1 Proposed Network Architecture

Neural networks designed for classification and trained in a discriminative manner are considered. Assume that the input data has non-negative values. This condition is often satisfied in practice. For example text documents in bag-of-words format or pixel intensities in images are naturally non-negative. Categorical data encoded using the 1-hot or thermometer-scale encoding is also non-negative and can be used. The desired output for each sample must be a unique class label.

The networks will have two layers. Each layer is determined by a matrix of weights and a vector of bias values, denoted by W_H and B_H for the hidden layer and by W_C and B_C for the classification (output) layer. For an input vector X the signal is propagated through the network according to the following relations. The hidden layer activations are $L_H(\boldsymbol{x}) = \sigma(\lambda(W_H \cdot \boldsymbol{x} + B_H))$, where $\sigma(\boldsymbol{x})$ denotes an element-wise application of the logistic sigmoid, $\sigma(x) = 1/(1+\exp(-x))$ and λ is a parameter denoting the neuron's gain. The classification layer activations are $L_C(\boldsymbol{x}) = \text{SoftMax}(W_C \cdot L_H(\boldsymbol{x}) + B_C)$, where the $\text{SoftMax}(\boldsymbol{v})$ function transforms a vector $\boldsymbol{v}$ into a vector of values according to $\text{SoftMax}(\boldsymbol{v})_i = \exp(\boldsymbol{v}_i)/\left(\sum_{j=1}^{J}\exp(\boldsymbol{v}_j)\right)$. The network assigns new data to the class represented by the output neuron with the highest activation value.

The optimization criterion used to train the network contains the log-likelihood and regularization terms. The output L_C of the softmax transfer function sums to 1 and can be treated as the a-posteriori probabilities of class labels given the input. According to this interpretation the network is trained by minimizing the negative log-likelihood of observing the given data set. Furthermore, the weights are regularized by minimizing their absolute values (ℓ_1 norm) and their squares (ℓ_2 norm) [28, 29], which is a penalty-based weight pruning mechanism. The combined action of the ℓ_1 and ℓ_2 penalties both selects important connections and limits their magnitude. Sparse activations of the hidden layer can additionally be enforced by minimizing the sum of the hidden neuron activations. This further enhances readability – proper classification of a sample must depend on only a few hidden neurons becoming active. The complete optimization target is:

$$\begin{aligned} \mathit{Loss} = & -\frac{1}{N}\sum_{s=1}^{N} \log\left(L_C(\boldsymbol{x}_s)_{\boldsymbol{y}_s}\right) \\ & + \sum_{i,j}\left(p_{H1}\left|W_{H\,i,j}\right| + p_{H2}W^2_{H\,i,j}\right) + \sum_{j,k}\left(p_{C1}\left|W_{C\,j,k}\right| + p_{C2}W^2_{C\,j,k}\right) \\ & + \frac{p_S}{N}\sum_{s=1}^{N} L_H(\boldsymbol{x}_s)_j \end{aligned} \tag{64}$$

where N is the number of samples, $(\boldsymbol{x}_s, \boldsymbol{y}_s)$ are individual data samples and p_{H1}, p_{H2}, p_{C1}, p_{C2}, p_S are regularization constants. The weight matrices W_H and W_C are constrained to contain only non-negative elements. The bias values are unconstrained and often negative.

Error backpropagation training with gradient descent of a network with the logistic sigmoid transfer function, that is used to keep all the signals non-negative, is numerically difficult [20]. Since the addition of weight constraints makes the optimization problem even harder, the L-BFGS-B second-order minimization algorithm [153] was chosen to train the network. When the training set was too large to be processed as a single batch, it has been divided into smaller subsets used for each epoch. The network was in those cases trained by executing a prescribed number of iterations during which the data were first randomly divided into batches of a few thousand samples, then a small number of steps of the L-BFGS-B algorithm was made on each batch to minimize (64). Stratified sampling was used to divide the data into batches to ensure that class label proportions are retained.

2 Expressive Power of Neural Networks with Non-negative Weights

It may seem that constraining the sign of weights of a neural network renders it too limited to be of any practical use. For instance, a neuron with positive weights and a logistic sigmoid activation function cannot logically invert its input. This section demonstrates by construction that a three layer network that uses the logistic sigmoid activation for hidden layers with the softmax activation function used for the output layer can shatter any given set of points.

The use of the softmax activation function is important for two reasons. First, it is the generalization of the cross-entropy error function to multiple classes and is more suited for a classification task. It can be derived from an assumption that the class labels are discrete and mutually exclusive. Then the outputs of the network using the softmax transfer function represent the a-posteriori class probabilities for a given sample. Second, the softmax function allows a degree of ambiguity for its inputs. A constant can be added to all inputs of a softmax function and the output will not be changed. This property is exploited in the proof that the networks with non-negative weights can learn any given labeling of data points. The key insight is that the network's decision is based on the maximally active output neuron. While it is not possible to lower the output of a neuron associated with a wrong class by setting its weight to be negative, it is possible instead increase the outputs of neurons for all other classes. Due to the ambiguity present in the softmax function, subtracting a number from a neuron's activation is mathematically equivalent to adding the same number to all other activations.

This property is also important for the understandability of the network. The softmax function necessitates that to lower the probability of a class all other outputs must be increased. This potentially produces large weight values. However, weight regularization terms penalize large values of weights and counteract the increase induced by the softmax activation. These two counterbalancing processes cause the output layer to become sparse, while hidden neurons learn concepts which relate to parts of characteristics that define an output class.

It will be shown that a three layer network can shatter every combination of points. First, note that adding a constant to all output weights does not change the value of the softmax function. Let the output layer compute the function:

$$L_C = \mathrm{SoftMax}(W \cdot L_H + B), \tag{65}$$

where L_C is the vector of classification (output) layer activations, W is the weight matrix, L_H is the vector of hidden layer activations, and B is the vector of bias values. Let $\mathbf{1}$ denote matrix whose all elements are 1 and let a be a constant. Then:

$$\begin{aligned} &\mathrm{SoftMax}\left((a\mathbf{1} + W)L_H + B\right) = \\ &\qquad \mathrm{SoftMax}(a\mathbf{1} \cdot L_H + W \cdot L_H + B). \end{aligned} \tag{66}$$

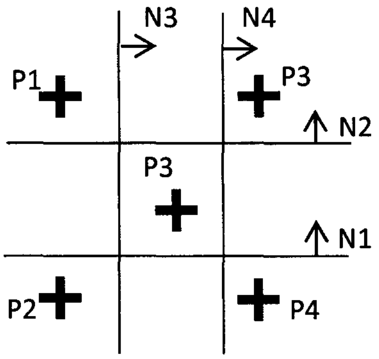
For simplicity, substitute $C = W \cdot L_H + B$. The product $\mathbf{1} \cdot L_H$ is a vector whose all elements are equal to the sum of L_H. It follows that:

$$\begin{aligned} \mathrm{SoftMax}(a\mathbf{1}L_H + C)_i &= \frac{\exp\left(\sum L_H\right)\exp(C_i)}{\sum_{j=1}^{n} \exp\left(\sum L_H\right)\exp(C_j)} \\ &= \mathrm{SoftMax}(C)_i \end{aligned} \tag{67}$$

Hence it is possible to transform any network with negative weights in the output layer into one with only non-negative weights by adding a large enough constant.

It is possible to construct a three layer (two hidden layers with sigmoid activation function followed by a classification layer with softmax activation) that will shatter a given set of N points described by their position in a k-dimensional space. Let $\boldsymbol{X} \in \mathbb{R}^{k \times N}$ be the data matrix. A network that computes any labeling of those points will be constructed. For simplicity assume that the gains in the logistic sigmoid transfer functions are infinite and the hidden neurons activations are always 0 or 1.

There are $O(kN)$ neurons in the first hidden layer. For every input dimension all data points are first projected onto this dimension. Then at most N threshold values between the projections are selected. Hidden neurons are added with a single nonzero weight equal to 1 corresponding to this dimension and bias equal to the threshold. Unless two columns of $\boldsymbol{X}$ are the same (i.e. two points are at exactly the same position), the activations H_1 of the first hidden layer are unique binary vectors.



	P1	P2	P3	P4	P5
N1	1	0	1	0	1
N2	1	0	0	0	1
N3	0	0	1	1	1
N4	0	0	0	1	1

Figure 19: Construction of a network with non-negative weights.

There are N neurons in the second hidden layer, one for each point. Their weights are equal, the weight $W_{i,j}$ connecting the j-th neuron in the first hidden layer to the i-th neuron in the second one equals to $2j$. Thus if the first hidden layer activations are treated as binary numbers, the products $W \cdot H_1$ are their decimal values. To every point corresponds one such number and the points can be ordered according to them. If the second hidden layer bias values are set to values in-between those numbers, the activations of the second hidden layer create a full-rank binary matrix (if the points are reordered, then for the n-th point the n first neurons are active, while the $N - n$ remaining ones are zero. Hence the activation matrix is triangular.) Thus output weights can be computed for every possible labeling of data points. In the last step a constant is added to output weights to ensure that all are non-negative.

3 Experimental Results

The proposed approach needs the specification of five parameters, four of which control the regularization and the last parameter, λ, sets the steepness of the sigmoid. The parameter λ was in all cases gradually increased to force the hidden neurons to operate in saturation. To determine the value of other parameters, the network was first trained without regularization. A regularization parameters were then selected by evaluating a few values that gave a similar value of the log-likelihood and regularization terms in (64). The Table 12 gives the values used

TABLE 12

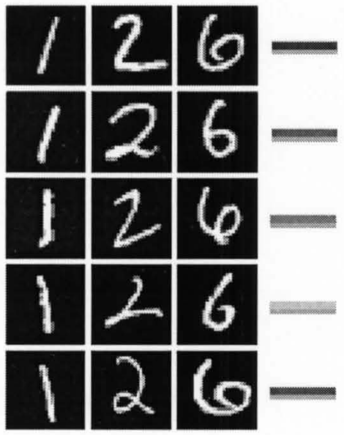
Values of regularization parameters used in the experiments.

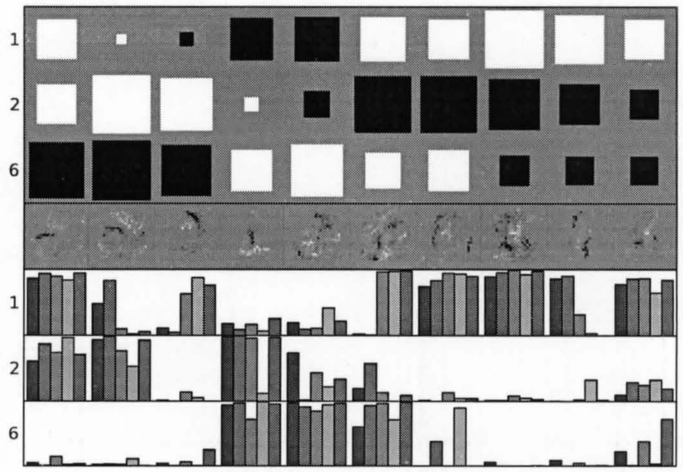
	Red. MNIST		Full MNIST		Reuters	
No. hidden	10	10	150	150	15	15
No. output	3	3	10	10	10	10
Non-neg?	F	T	F	T	F	T
λ	Annealed exponentially from 1.0 to 2.0					
p_{H1}	1e-4	3e-4	0	1e-5	1e-4	1e-4
p_{H2}	0	0	0	0	0	0
p_{C1}	0	0	0	1e-4	3e-4	3e-4
p_{C2}	1e-4	1e-4	1e-4	3e-6	3e-4	3e-4
p_S	0	0	0	1e-3	0	0

for the experiments.

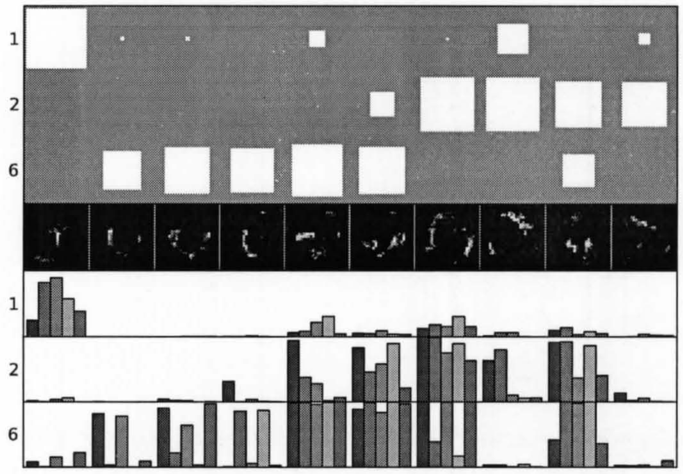
In the first experiment networks constructed with and without non-negativity weight constraints were compared on a subset of the MNIST handwritten digit data limited to digits 1, 2, and 6. The full MNIST data set contains 60000 training and 10000 testing grayscale images of handwritten digits which were scaled and centred inside a 28x28 pixel box. It can be obtained along with a summary of classifiers' accuracies from `http://yann.lecun.com/exdb/mnist/index.html`. Figure 20 presents a selection of test patterns and the weights of the two networks. An immediate consequence of the non-negativity constraints is sparsification of weights in the classification layer. Furthermore, the patterns learned by the hidden neurons allow easy interpretation. They are localized and tend to look like parts of digits (e.g. neurons 2-4 look like the rounded bottom of digit 6). In contrast, the hidden neurons of the unconstrained network are less localized. They contain both positive and negative weights covering most of the input image, which makes it harder to visualize to what patterns they respond. The bar charts indicate the activations of hidden neurons for the sample input patterns. It can be seen that neurons in both networks discriminate between digits and tend to work in the nonlinear parts of their activation functions, resembling threshold gates. The unconstrained network is more accurate and achieves 1% error rate, compared with 1.5% for the constrained one. In general, the trend was observed that more understandable networks show lower accuracy. However, in certain situations a better insight into the data outweighs the benefits of an accurate but opaque classifier.

In the next large-scale experiment, we used the full MNIST data to build a constrained and unconstrained neural network with 150 hidden neurons and 10 outputs. We compare in Figure 23 the depictions of weights of 32 randomly selected hidden neurons with 32 features obtained with PCA and with 32 obtained with NMF. Full networks are shown in the Figure 21 and 22. The unconstrained network shows a much lower error rate of 2.4%, compared with 4.9% for the constrained one. To put those numbers into perspective, state-of-the-art 1998 error rate on the MNIST for a two layer neural network was 4.7%. Once again, the non-negativity constraints result in the emergence of sparser and more localized weight distributions of the hidden neurons, which often filter distinctive parts of digits. In contrast, the hidden neurons of the unconstrained network react to whole pictures, thus it is difficult to estimate intuitively their influence on the classifier's output. Similarly, the patterns learned by PCA are holistic, non-localized ones. But for the first few, it is hard to describe their contents. It is also difficult to see how they relate to the shapes of different digits. Further, the NMF has learned sparse, localized, and interpretable features. However, only a few patterns resemble parts of digits, like the vertical bar. Most of the features seem to down-sample input images on a non-uniform grid and do not provide cues for classification. This is caused by two factors. First, unlike the neural network with non-negative constraints on weights, the NMF model does not aim at class discrimination. Second, NMF imposes no limits on the number of features activated by a sample. Increasing the rank of factorization (the total number of features) only worsens the issue, as in the limit each NMF feature will be a single pixel and the NMF-transformed data will contain no new information. On the other hand, decreasing the rank leads the NMF features to look like blurred shapes of the simplest digits. The addition of a constraint on the number of coactive features, while allowing a large total number of features, has been shown to promote the learning of a parts-based decomposition [58, 60, 154]. This is because, in contrast to a limited-rank decomposition, a large dictionary of features is


(a)



(b)



(c)

Figure 20: (a) Exemplary digits from the MNIST dataset. The weights of a network trained (b) without constraints and (c) with non-negative constraints. The weights of the classification (output) layer are plotted as a diagram with one row for each output neuron and one column for every hidden (input) neuron. The area of each square is proportional to the weight's magnitude; white indicates positive and black negative sign. Below each column of the diagram, the weights of hidden neurons are printed as an image. The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with, the value 0 corresponding to gray in (b) and to black in (c). The biases are not shown. The hidden neurons have been rearranged for better presentation. The bar charts at the bottom of the plots show the activation of hidden neurons for the digits presented in (a). Each row depicts the activations of each hidden neuron for five color-coded examples of the same digit.

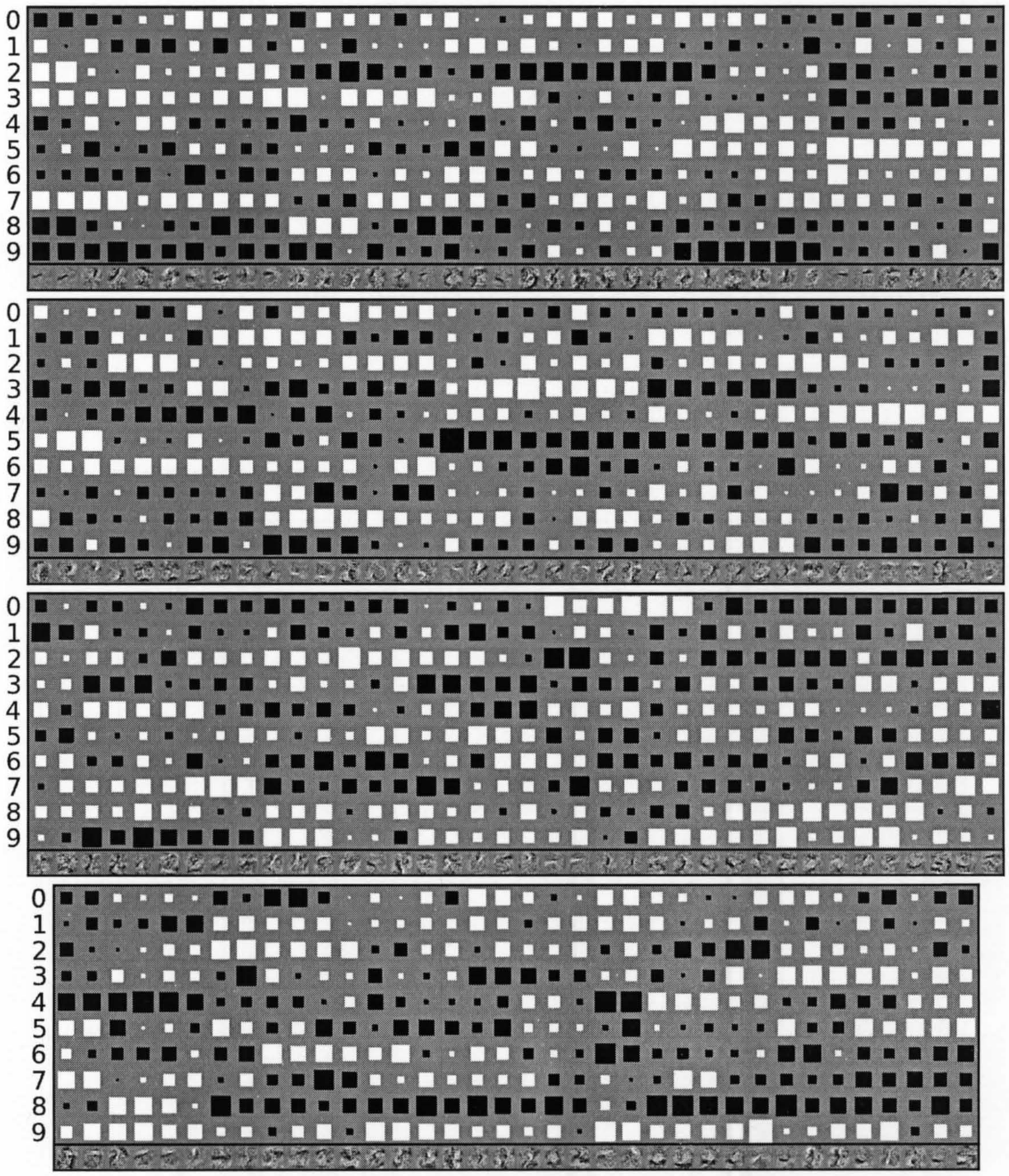


Figure 21: The weights of a network trained on the full MNIST dataset without weight constraints. The weights of the classification (output) layer are plotted as a diagram with one row for each output neuron and one column for every hidden (input) neuron. The area of each square is proportional to the weight's magnitude; white indicates positive and black negative sign. Below each column of the diagram, the weights of hidden neurons are printed as an image. The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with, the value 0 corresponding to gray. The biases are not shown. The hidden neurons have been rearranged for better presentation.

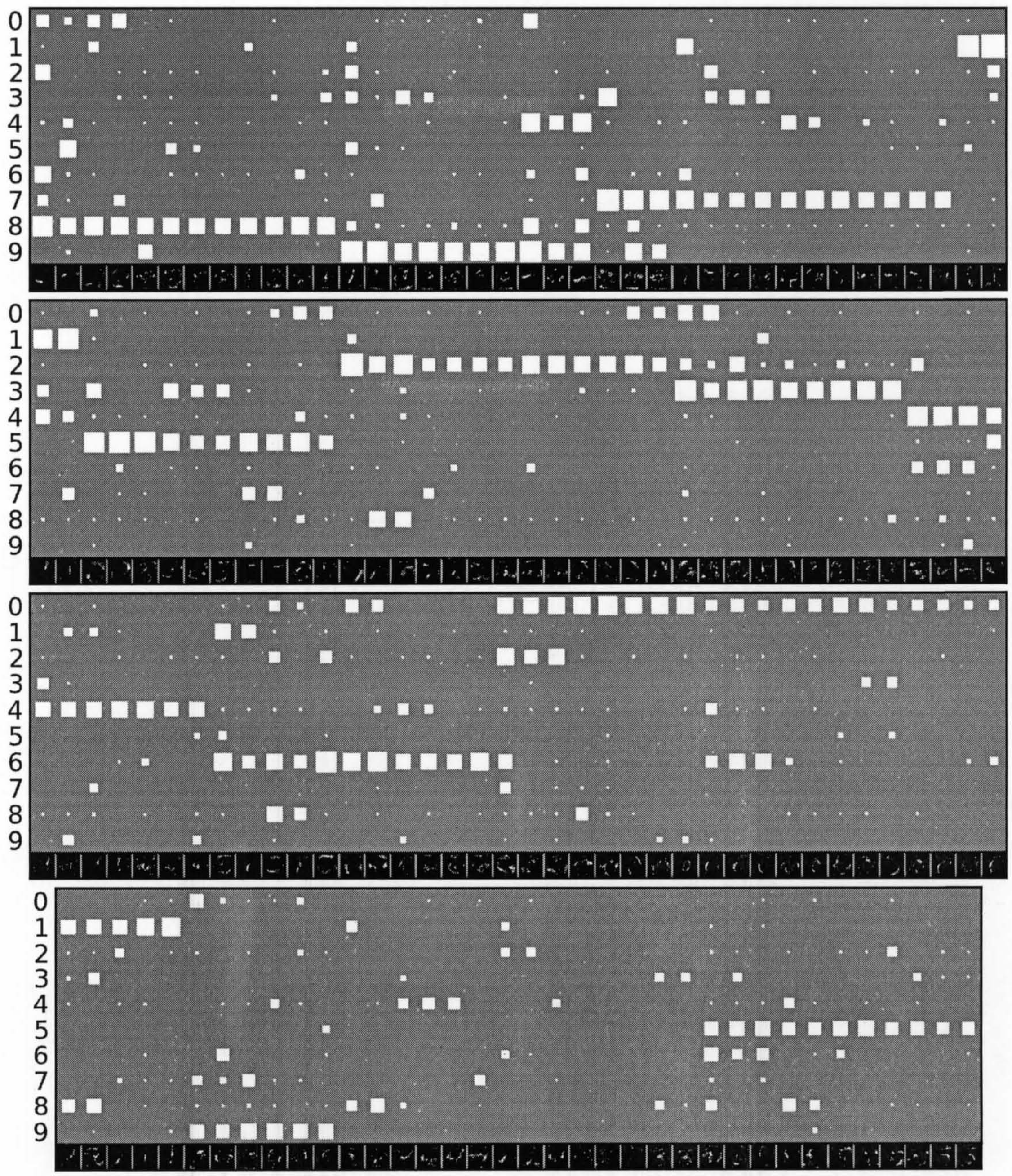


Figure 22: The weights of a network trained on the full MNIST dataset with non-negativity weight constraints. The weights of the classification (output) layer are plotted as a diagram with one row for each output neuron and one column for every hidden (input) neuron. The area of each square is proportional to the weight's magnitude. Below each column of the diagram, the weights of hidden neurons are printed as an image. The intensity of each pixel is proportional to the magnitude of the weight connected to that pixel in the input image with, the value 0 corresponding to black. The biases are not shown. The hidden neurons have been rearranged for better presentation.

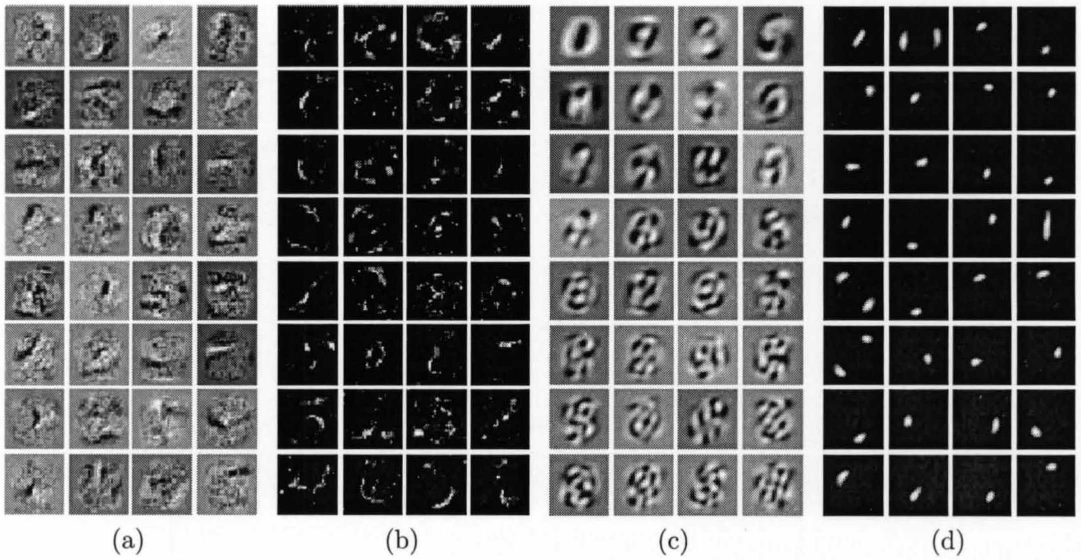


Figure 23: Weights of randomly selected 32 out of 150 hidden neurons of unconstrained network (a) and network with weigth non-negativity constraints (b). 32 first principal components (c). 32 filters learned by NMF (d).

created. The features, in turn, must be complex enough to provide adequate input reconstruction from just the few active ones.

In the last experiment networks were compared on the Reuters-21578 text categorization collection. It is composed of documents that appeared in the Reuters newswire in 1987. The ModApte split limited to ten most frequent categories was used. The processed (stemming, stop-word removal) version in bag-of-words format obtained from `http://people.kyb.tuebingen.mpg.de/pgehler/rap/` was used. This dataset is challenging because the borders between topics are fuzzy and documents may belong to many categories simultaneously. During training such documents were used with all possible labels. For testing a document was counted as correctly classified when the network assigned it to one of the classes to which it belonged. The networks had 15 hidden and 10 output neurons (one for each category). The unconstrained network is slightly more accurate and achieves an error rate of 12.4%, compared with 12.8% for the constrained one. The weights of the two networks are portrayed in Figure 24. An interpretation of the hidden neurons is provided by listing words associated with the strongest weights. The word "blah" has no meaning and is artificially added noise. The non-negative network has been observed to be more sensitive to it, as many hidden neurons react to it. The neurons in the unconstrained network seem to convey meaning by being both active and inactive, because the words associated with positive and negative weights fall into distinct categories. Furthermore, the matrix of output weights is dense and difficult to interpret. On the other hand, the output weights of the non-negative network are sparse and allow for an interpretation of relations between topics. The closeness of topics "corn", "grain", and "wheat" is detected as the weights for those categories form a cluster. The topic "trade" is linked to categories describing goods that can be traded. The words listed for hidden neurons corroborate those interpretations, e.g. the neuron reacting to words "trade", "rate", "fed", "dollar" is linked to topics "money-fx" (foreign exchange), "interest", and "trade".

4 Conclusions

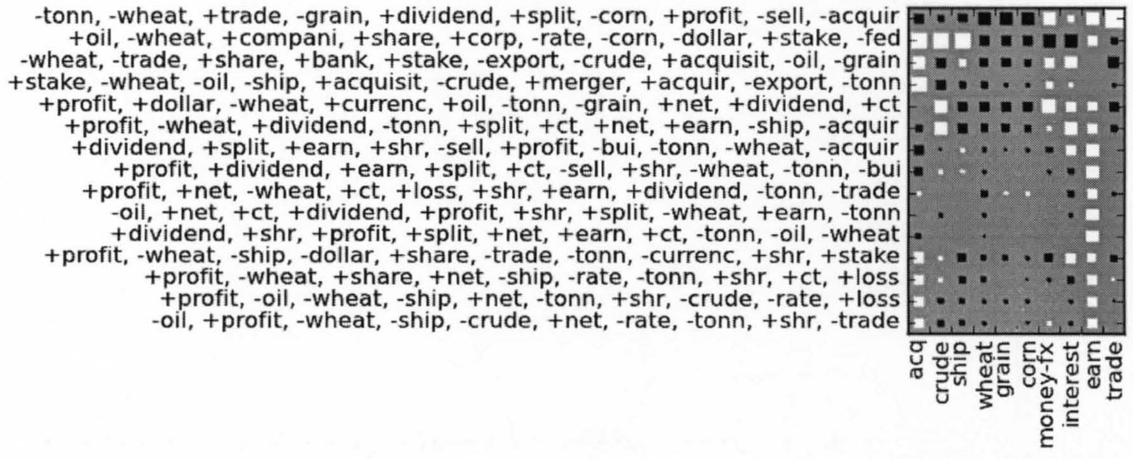
It was demonstrated how constraining the weights of a neural network to be non-negative improves network understandability and leads to intuitively understandable hidden neurons. To the best of our knowledge, this is the first attempt at discriminative training of understandable neural networks on large, nontrivial datasets.

Deriving understandable descriptions of observations is the hallmark of human intelligence. The presented approach is but a single step on the road towards pattern recognition tools that help not only to make predictions about data, but also empower their user with new insights and concepts derived from that data.

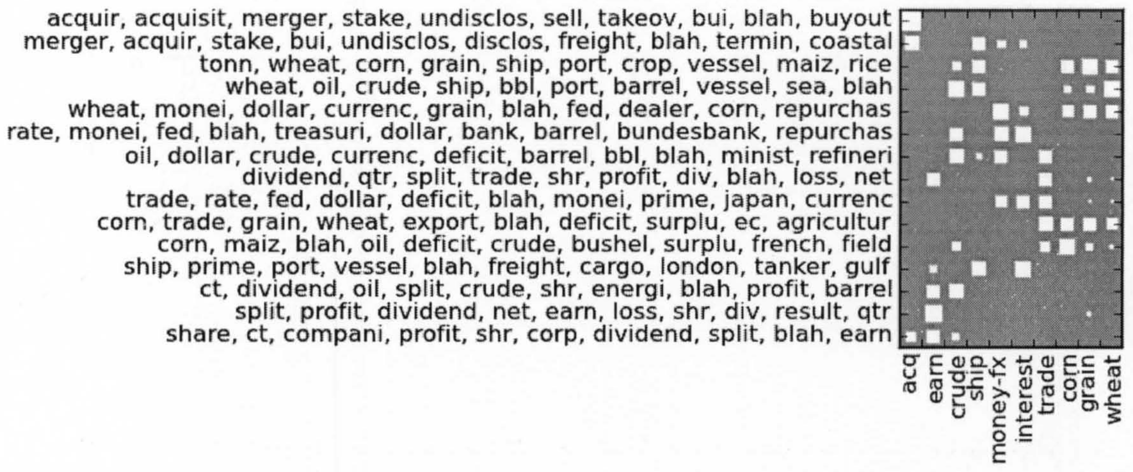
B Robust Sparse Coding by Minimizing an L1-L1 Problem[2]

The sparse coding problem consists of finding a vector of feature coefficients $\boldsymbol{v} \in \mathbb{R}^p$ that express an input vector $\boldsymbol{x} \in \mathbb{R}^k$ as a linear combination of basis vectors

[2]This section is based on [155] (J. Chorowski and J. Zurada, "Obtaining full regularization paths for robust sparse coding with applications to face recognition." in *Accepted to International Conference on Machine Learing and Applications – ICMLA*, 2012).



(a)



(b)

Figure 24: Networks trained on the Reuters-21578 data: with unconstrained weights (a) and with non-negative weight constraints (b). Input neurons are characterized by listing ten words connected to weights having large absolute value. The + and - signs indicate the sign of the weight in (a). Each column of the diagram depicts weights of an output neuron, the size varies with weight value and black or white filling indicates sign as in Figure 20. The neurons have been rearranged for better presentation.

that form a fixed dictionary $\boldsymbol{D} \in \mathbb{R}^{k \times p}$. The vector of reconstruction coefficients $\boldsymbol{v}$ is required to be sparse, i.e. to contain few non-zero elements. Since the minimization of the number of nonzero elements in $\boldsymbol{v}$ is a difficult combinatorial problem, practical implementations resolve to minimize the ℓ_1 norm of $\boldsymbol{v}$ instead:

$$\min_{\boldsymbol{v}} ||\boldsymbol{v}||_1 \text{ subject to: } \boldsymbol{x} = \boldsymbol{D}\boldsymbol{v}. \tag{68}$$

However, a perfect reconstruction is often impossible to obtain and the problem is relaxed to allow approximations to $\boldsymbol{x}$:

$$\min_{\boldsymbol{v}} ||\boldsymbol{v}||_1 \text{ subject to: } ||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{v}||_2 \leq \epsilon. \tag{69}$$

Alternatively, the following unconstrained problem may be solved with the parameter λ balancing the reconstruction error and sparsity of coefficients $\boldsymbol{v}$:

$$\min_{\boldsymbol{v}} 0.5||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{v}||_2^2 + \lambda||\boldsymbol{v}||_1. \tag{70}$$

This formulation of sparse coding is equivalent to LASSO regression [156]. Recently, the LARS algorithm has been proposed for efficient computation of the path traced by coefficients $\boldsymbol{v}$ when the regularization parameter λ is varied [157, 158]. The LARS algorithm has been extended to piecewise-quadratic error measures in [159]. The key aspect utilized by those methods is that the values of coefficients $\boldsymbol{v}$ are piecewise-linear with regard to the regularization constant λ. Many other optimization problems that incorporate sparsity constraints can be solved by tracing the path of the reconstruction coefficients. An approximate algorithm that is applicable to training generalized linear models is presented in [160].

In the robust sparse coding problem the ℓ_1 norm is used both to measure the reconstruction error and to regularize the coefficients:

$$\min_{\boldsymbol{v}} ||\boldsymbol{D}\boldsymbol{v} - \boldsymbol{x}||_1 \text{ subject to: } ||\boldsymbol{v}||_1 \leq \tau \tag{71}$$

The more frequently used ℓ_2 norm is sensitive to outliers because a large error in a single component dominates the norm. In contrast, the ℓ_1 norm is more robust because large errors are not magnified by taking their squares. Under a Bayesian view, the ℓ_2 norm assumes that errors are normally distributed, while the ℓ_1 norm assumes a doubly exponential (Laplace) distribution. It has heavier tails than the Gaussian, which lessens outliers influence on the location of the mean. Many practical applications, such as occluded face recognition, require the robustness to outliers and solve the problem (71) [161–163].

Robust sparse coding (71) can readily be solved for a fixed value of τ using linear programming techniques. In this contribution an algorithm is developed that efficiently computes the path traced by coefficients $\boldsymbol{v}(\tau)$ when the regularization constant τ is varied. It has been shown in [164] that when ℓ_1 regularization is applied to weights of a Support Vector Machine, the path traced by the weights the weights is piecewise-linear with respect to the sum of weights' absolute values. Moreover, it has been stated in [159] that solutions of a more general class of loss functions, which includes the ℓ_1 norm, paired with the ℓ_1 coefficient regularization also yield piecewise-linear regularization paths. While the proposed algorithm is inspired by the L1-SVM method, the discussion of KKT optimality conditions is novel and provides important computation time savings. Furthermore, subgradient calculus is used to simplify the derivation.

1 Obtaining Full Regularization Paths

The problem of obtaining regularization paths for the general problems will be analyzed:

$$\min \sum_{j=1}^{k} l(R_j) \text{ subject to: } ||\boldsymbol{v}||_1 \leq \tau, \tag{72}$$

and to its equivalent formulation:

$$\min \sum_{j=1}^{k} l(R_j) + \lambda ||\boldsymbol{v}||_1, \tag{73}$$

where $R_j = \boldsymbol{D}_{j:}\boldsymbol{v} - \boldsymbol{x}$ is the j-th reconstruction residual and $\boldsymbol{D}_{j:}$ indicates the j-th row of $\boldsymbol{D}$. The problems (72) and (73) are equivalent because they share the same Lagrangian. Three cases of the penalty function l can be considered: the ℓ_2 norm based penalty $l_2(R) = 0.5R^2$, the ℓ_1 norm based penalty $l_1(R) = |R|$, and the modified Huber penalty $l_H(R) = R^2/(2\delta)$ for $|R| \leq \delta$, $(|R| - \delta/2)$ otherwise. The Huber penalty can be interpreted as a differentiable approximation to $l_1$penalty with the parameter δ controlling the radius of the quadratic region around $R = 0$, the singularity of $|R|$. The l_2 and l_H penalties are presented based on [159] for the completeness of discourse. The in-depth analysis of the case of the l_1 penalty is the main result of this contribution.

Subgradient Primer

The problems analyzed in this contribution are non-differentiable. They can be essentially dealt with in two ways: by defining artificial constrained variables to transform the problems into smooth, but much larger ones or by using subgradient methods. The later solution greatly simplifies the analysis of the case of l_1 penalty function and necessary theorems and definitions are introduced here after [165–167].

Let f be a convex function. A vector g is called a *subgradient* of f at point $x_0 \in \operatorname{dom} f$ if for any $x \in \operatorname{dom} f$ it holds that: $f(x) \geq f(x_0) + g^T \cdot (x - x_0)$. The set of all subgradients of f at x_0, $\partial f(x_0)$ is called the *subdifferential* of function f at point x_0. In example, for $f(x) = |x|$, the subdifferential at $x \neq 0$ is $\operatorname{sgn}(x)$, and at $x = 0$ it is the whole segment $[-1, 1]$. There are two important theorems about subgradients that will allow reasoning about optimality of solutions:

1. Unconstrained optimality condition: $f(x^*) = min_{x \in \operatorname{dom} f} f(x)$ if and only if $0 \in \partial f(x^*)$ [165, 167].

2. KKT constrained optimality conditions. Consider:

$$\min f_0(x) \ \text{subject to:} \ f_i(x) \leq 0, \ i = 1, \ldots, m. \tag{74}$$

   If for $i = 0, 1, \ldots, m$ f_i are convex and defined on $\mathbb{R}^n$ and the problem is strictly feasible, then x^* is optimal if and only if: $0 \in \partial f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \partial f_i(x^*)$, and for all $i = 1, \ldots, m$ $f_i(x^*) \leq 0$, $\lambda_i^* \geq 0$, $\lambda_i^* f_i(X^*) = 0$ [166, 167].

Differentiable Penalty Functions

Similarly to [159] the optimality conditions of (73) will be analyzed. Unlike [159] subgradients will be used to simplify the arguments. Let $L = \sum_{j=1}^{k} l(R_j)$ be the total loss. At points where L is differentiable the following conditions hold:

$$0 = \frac{\partial L}{\partial \boldsymbol{v}_i} + \lambda \cdot \operatorname{sgn}(\boldsymbol{v}_i) \text{ if } \boldsymbol{v}_i \neq 0 \tag{75a}$$

$$0 \in \frac{\partial L}{\partial \boldsymbol{v}_i} + \lambda \cdot [-1, 1] \text{ if } \boldsymbol{v}_i = 0. \tag{75b}$$

Let $\mathcal{A}$ be the set of indexes of nonzero coefficients $\boldsymbol{v}$ called the active set $\mathcal{A} = \{i : \boldsymbol{v}_i \neq 0\}$. Note that for $i \in \mathcal{A}$ it is necessary that $\left|\frac{\partial L}{\partial \boldsymbol{v}_i}\right| = \lambda$. It follows that $i \notin \mathcal{A} \implies \left|\frac{\partial L}{\partial \boldsymbol{v}_i}\right| \leq \lambda$. When the loss L is twice differentiable and the penalty l is right differentiable one can differentiate (75a) to obtain [159]:

$$\frac{\partial \boldsymbol{v}_{\mathcal{A}}}{\partial \lambda} = -(\nabla^2 L_{\mathcal{A}})^{-1} \operatorname{sgn}(\boldsymbol{v}_{\mathcal{A}}), \tag{76}$$

where $\nabla^2 L_{\mathcal{A}}$ is the Hessian matrix of the loss with regard to the active coefficients and $\boldsymbol{v}_{\mathcal{A}}$ is the vector of active coefficients only. In the case of l_2 and l_H $\frac{\partial \boldsymbol{v}_{\mathcal{A}}}{\partial \lambda}$ is piecewise constant [159]. The LARS algorithm uses this property and iterates over "events":

- variable $\boldsymbol{v}_i = 0$ enters the active set when: $\left|\frac{\partial L}{\partial \boldsymbol{v}_i}(\boldsymbol{v}_i(\lambda))\right| = \lambda$;
- variable $\boldsymbol{v}_i \neq 0$ leaves the active set when $\boldsymbol{v}_i(\lambda) = 0$;
- a knot of l is crossed, for the Huber loss these occur when $R_j(\lambda) = \pm\delta$.

In this way the whole solution path with regard to λ is obtained.

Non-differentiable Penalty Function

In the case of the l_1 penalty function the solution is not piecewise linear with regard to the lambda parameter, but with regard to $\tau = ||\boldsymbol{v}||_1$. Furthermore, a different strategy is needed to determining the gradient $\frac{\partial \boldsymbol{v}}{\partial \tau}$. The solution is based on the L1-SVM algorithm [164]. Consider the minimization problem (71). Let $R_j = \boldsymbol{D}_{j:}\boldsymbol{v} - \boldsymbol{x}_j$ be the j-th residue and let $L = ||R||_1$ be the total loss. The Lagrangian is

$$\mathcal{L}(\boldsymbol{v}, \lambda) = L + \lambda(||\boldsymbol{v}||_1 - \tau) = ||\boldsymbol{D}\boldsymbol{v} - \boldsymbol{x}||_1 + \lambda(||\boldsymbol{v}||_1 - \tau). \tag{77}$$

Optimal points need to satisfy the conditions:

$$\tau \geq ||\boldsymbol{v}||_1 \tag{78a}$$

$$0 = \sum_{j:R_j \neq 0} \boldsymbol{D}_{ji} \operatorname{sgn}(R_j) + \sum_{j:R_j = 0} \boldsymbol{D}_{ji} \frac{\partial |R_j|}{\partial R_j} + \lambda \begin{cases} \operatorname{sgn}(\boldsymbol{v}_i) & \text{if } \boldsymbol{v}_i \neq 0 \\ \frac{\partial |\boldsymbol{v}_i|}{\partial \boldsymbol{v}_i} & \text{otherwise.} \end{cases} \tag{78b}$$

$$0 = \lambda(||\boldsymbol{v}||_1 - \tau) \tag{78c}$$

$$\lambda \geq 0, \tag{78d}$$

where $\frac{\partial |R_j|}{\partial R_j} \in [-1, 1]$ and $\frac{\partial |\boldsymbol{v}_i|}{\partial \boldsymbol{v}_i} \in [-1, 1]$ are values of subdifferentials for which the optimality condition holds.

It is now shown how to determine the gradient $\frac{\partial \boldsymbol{v}_i}{\partial \tau}$. Suppose that for a $\hat{\tau}$ known are the optimum coefficients $\hat{\boldsymbol{v}}$ which yield residues $\hat{R}$. Define $\boldsymbol{\beta}_i = \frac{\partial \boldsymbol{v}_i}{\partial \tau}$ to be the right derivative of the coefficients $\boldsymbol{v}$ with regard to the regularization parameter τ. The vector $\boldsymbol{\beta}$ must ensure that the point $\boldsymbol{v} = \hat{\boldsymbol{v}} + s\boldsymbol{\beta}$ is the optimum for $\tau = \hat{\tau} + s$. When the step s is sufficiently small, the nonzero residues and nonzero coefficients will not become zero or change sign. Hence $\boldsymbol{\beta}$ can be determined as the solution of another minimization problem:

$$\min_{\boldsymbol{\beta}} ||\boldsymbol{D}\boldsymbol{v} - \boldsymbol{x}||_1 \text{ subject to: } ||\boldsymbol{v}||_1 \leq \tau. \tag{79}$$

Substitute the known signs of nonzero residues and coefficients:

$$\begin{gathered} \min_{\boldsymbol{\beta}} \sum_{j:\hat{R}_j \neq 0} \operatorname{sgn}(\hat{R}_j)(\boldsymbol{D}_{j:}\hat{\boldsymbol{v}} - y_j + s\boldsymbol{D}_{j:}\boldsymbol{\beta}) + \sum_{j:\hat{R}_j = 0} s|\boldsymbol{D}_{j:}\boldsymbol{\beta}| \\ \text{subject to:} \\ \sum_{i:\hat{\boldsymbol{v}}_i \neq 0} \operatorname{sgn}(\hat{\boldsymbol{v}}_i)(\hat{\boldsymbol{v}}_i + s\boldsymbol{\beta}_i) + \sum_{i:\hat{\boldsymbol{v}}_i = 0} s|\boldsymbol{\beta}_i| \leq \hat{\tau} + s. \end{gathered} \tag{80}$$

The problem is simplified by subtracting constant terms from the objective function, subtracting $\hat{\tau}$ from the constraint and by dividing both the objective and

Algorithm 9: Obtaining full solution path of problem (71).

```
1: Start with $\tau^0 = 0$, $\boldsymbol{v}^0 = 0$, $R^0 = -\boldsymbol{x}$.
2: Solve (81) to get $\boldsymbol{\beta}^0$.
3: while The target of (71) decreases do
4:     Make step $s^m$ in direction $\boldsymbol{\beta}^m$ choosing the smallest step at which
       a coefficient $\boldsymbol{v}_i$ becomes 0, or a residue $R_j$ becomes 0.
5:     Solve (81) to get $\boldsymbol{\beta}^{m+1}$.
6: end while
```

constraint by s to obtain:

$$\begin{gathered}\min_{\boldsymbol{\beta}} \sum_{j:\hat{R}_j \neq 0} \operatorname{sgn}(\hat{R}_j)\boldsymbol{D}_{j:}\boldsymbol{\beta} + \sum_{j:\hat{R}_j = 0} |\boldsymbol{D}_{j:}\boldsymbol{\beta}| \\ \text{subject to:} \\ \sum_{i:\hat{\boldsymbol{v}}_i \neq 0} \operatorname{sgn}(\hat{\boldsymbol{v}}_i)\boldsymbol{\beta}_i + \sum_{i:\hat{\boldsymbol{v}}_i = 0} |\boldsymbol{\beta}_i| \leq 1.\end{gathered} \tag{81}$$

Note that $\boldsymbol{\beta}$ are piecewise-constant with respect to τ. Hence the coefficients $\boldsymbol{v}$ are piecewise-linear with respect to τ. The coefficients $\boldsymbol{v}$ are not unique for a value of λ, which is piecewise constant with discontinuities at points where a coefficient or residue becomes zero or changes sign.

It is now possible to formulate the Algorithm 9 that traces a full solution path of problem (71). The crucial question is how to solve (81) efficiently in line 5 of the algorithm loop. The KKT conditions for optimality will again be analyzed. To simplify the conditions first observe that $R_j = \hat{R}_j + \boldsymbol{D}_{j:}\boldsymbol{\beta} = \boldsymbol{D}_{j:}\boldsymbol{\beta}$ if $\hat{R}_j = 0$. From the KKT optimality conditions it is that:

$$\sum_{i:\hat{\boldsymbol{v}}_i \neq 0} \operatorname{sgn}(\hat{\boldsymbol{v}}_i)(\boldsymbol{\beta}_i) + \sum_{i:\hat{\boldsymbol{v}}_i = 0} |\boldsymbol{\beta}_i| \leq 1 \tag{82a}$$

$$0 = \sum_{j:\hat{R}_j \neq 0} \operatorname{sgn}(\hat{R}_j)\boldsymbol{D}_{ji} + \sum_{j:\hat{R}_j = 0} \frac{\partial |R_j|}{\partial R_j}\boldsymbol{D}_{ji} + \lambda \operatorname{sgn}(\hat{\boldsymbol{v}}_i) \text{ for } \hat{\boldsymbol{v}}_i \neq 0 \tag{82b}$$

$$0 = \sum_{j:\hat{R}_j \neq 0} \operatorname{sgn}(\hat{R}_j)\boldsymbol{D}_{ji} + \sum_{j:\hat{R}_j = 0} \frac{\partial |R_j|}{\partial R_j}\boldsymbol{D}_{ji} + \lambda \frac{\partial |\boldsymbol{\beta}_i|}{\boldsymbol{\beta}_i} \text{ for } \hat{\boldsymbol{v}}_i = 0 \tag{82c}$$

$$\lambda\Big(\sum_{i:\hat{\boldsymbol{v}}_i \neq 0} \operatorname{sgn}(\hat{\boldsymbol{v}}_i)(\boldsymbol{\beta}_i) + \sum_{i:\hat{\boldsymbol{v}}_i = 0} |\boldsymbol{\beta}_i| - 1\Big) = 0 \tag{82d}$$

$$\lambda \geq 0, \tag{82e}$$

where $\frac{\partial|R_j|}{\partial R_j} \in [-1,1]$ and $\frac{\partial|\beta_i|}{\beta_i} \in [-1,1]$. λ is the Lagrange multiplier.

Two comments are necessary. First, λ was used to denote the Lagrange multiplier in (82d) because its value is the same as the value of the Lagrange multiplier of problem (71). It is so because the target and constrains were scaled by the same nonnegative factor, i.e. by s. Second, multiplying (82b) and (82c) by the respective elements of $\boldsymbol{\beta}$ and summing yields:

$$\begin{aligned}\lambda &= -\sum_{j:\hat{R}_j\neq 0} \operatorname{sgn}(\hat{R}_j)\boldsymbol{D}_{j:}\boldsymbol{\beta} + \sum_{j:\hat{R}_j=0} \frac{\partial|\boldsymbol{D}_{j:}\boldsymbol{\beta}|}{\partial \boldsymbol{D}_{j:}\boldsymbol{\beta}}\boldsymbol{D}_{j:}\boldsymbol{\beta} \\ &= -\sum_{j:\hat{R}_j\neq 0} \operatorname{sgn}(\hat{R}_j)\boldsymbol{D}_{j:}\boldsymbol{\beta} + \sum_{j:\hat{R}_j=0} |\boldsymbol{D}_{j:}\boldsymbol{\beta}|,\end{aligned} \tag{83}$$

which means that the optimal value of (81) is equal to $-\lambda$.

Denote the set of indexes of active coefficients at $\hat{\tau}$ by $\mathcal{A} = \{i : \hat{\boldsymbol{v}}_i \neq 0\}$, the set of indexes of satisfied residues $\mathcal{S} = \{j : \hat{R}_j = 0\}$, and the set of not satisfied residues by $\mathcal{N} = \mathcal{S}^{\mathrm{C}}$ When progress is possible, (82a) is active and λ is nonzero. Furthermore, the active $\hat{\boldsymbol{v}}$ yield a system of $|\mathcal{A}|$ linear equations with $|\mathcal{S}| + 1$ unknowns which are the dual variable λ and the values of the subdifferentials $\frac{\partial|R_j|}{\partial R_j}$. If it is assumed that the system has a unique solution, it can obtained either when the number of equations is increased or when the number of variables is decreased. The first corresponds to the inclusion of an inactive coefficient into the active set. The second corresponds to the assumption that a satisfied residual will become nonzero. Furthermore, it is possible to verify the optimality of any solution of (81) by asserting that $\lambda > 0$, $\frac{\partial|R_j|}{\partial R_j} \in [-1,1]$, and $\frac{\partial|\beta_i|}{\beta_i} \in [-1,1]$. To obtain the values of $\boldsymbol{\beta}$ the equation (82a) is used along with equations $R_j = 0 \implies D_{j:}\boldsymbol{\beta} = 0$ for residues that will stay satisfied.

It is now possible to provide the full algorithm for solving (81) under the assumption that it has a unique solution:

1. Try to solve (81) by removing a residual from the satisfied set. Check all cases by solving for all $n \in \mathcal{S}$:

$$\begin{bmatrix} 0_{|\mathcal{S}|-1} \\ 1 \end{bmatrix} = \begin{bmatrix} D_{(\mathcal{S}\setminus\{n\})\mathcal{A}} \\ \operatorname{sgn} \boldsymbol{v}_A^T \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\beta}_{\mathcal{A}} \end{bmatrix}, \tag{84}$$

where $0_{|\mathcal{S}|-1}$ is a vector of $|\mathcal{A}|-1$ zeros. The optimality of the solution can be verified by solving:

$$\begin{bmatrix} -\boldsymbol{D}^T_{\mathcal{N}\mathcal{A}} \operatorname{sgn} \hat{R}_{\mathcal{N}} - \boldsymbol{D}^T_{n\mathcal{A}} \operatorname{sgn} R_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{D}^T_{(\mathcal{S}\setminus\{n\})\mathcal{A}} & \operatorname{sgn} \boldsymbol{v}_{\mathcal{A}} \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial |R_{(\mathcal{S}\setminus\{n\})}|}{\partial R_{(\mathcal{S}\setminus\{n\})}} \\ \lambda \end{bmatrix}. \tag{85}$$

Note that the two systems of equations are closely related because they are defined by the same matrix (up to transposition).

2. Try to add a new variable to the active set by solving for all $r \notin \mathcal{A}$:

$$\begin{bmatrix} 0_{\mathcal{A}} \\ 1 \end{bmatrix} = \begin{bmatrix} D_{\mathcal{S}\mathcal{A}} & D_{\mathcal{S}r} \\ \operatorname{sgn} \boldsymbol{v}^T_{\mathcal{A}} & \operatorname{sgn} \boldsymbol{\beta}_r \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{\beta}_{\mathcal{A}} \\ \boldsymbol{\beta}_r \end{bmatrix}, \tag{86}$$

where $0_{\mathcal{A}}$ is a vector of $|\mathcal{A}|$ zeros. The optimality of the solution can be verified by solving:

$$\begin{bmatrix} -\boldsymbol{D}^T_{\mathcal{N}\mathcal{A}} \operatorname{sgn} \hat{R}_{\mathcal{N}} \\ -\boldsymbol{D}^T_{\mathcal{N}r} \operatorname{sgn} \hat{R}_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{D}^T_{\mathcal{S}\mathcal{A}} & \operatorname{sgn} \boldsymbol{v}_{\mathcal{A}} \\ \boldsymbol{D}^T_{\mathcal{S}r} & \operatorname{sgn} \boldsymbol{\beta}_r \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial |R_{\mathcal{S}}|}{\partial R_{\mathcal{S}}} \\ \lambda \end{bmatrix}. \tag{87}$$

Again the two systems of equations are defined by essentially the same matrix.

3. The optimum solution can be selected either by verifying the optimality conditions for each considered case, or by computing target values and picking the point yielding the lowest target value.

2 Runtime Complexity Considerations

The running time of described algorithms depends on the product of two factors: how many events occur along the path and how expensive it is to process a single event. k will be used to denote the dimensionality of a single sample $\boldsymbol{x}$ and p to denote the number of basis (columns) in $\boldsymbol{D}$. In other words, $\boldsymbol{D}$ is an $k \times p$ matrix.

In the case of the l_2 or the l_H loss functions, during a single event the gradient $\frac{\partial L}{\partial \boldsymbol{v}}$ needs to be computed in $O(kp)$ operations. Choosing the step length also requires $O(kp)$ operations. Then, the new direction $\frac{\partial \boldsymbol{v}}{\partial \lambda}$ must be computed using (76). It can be performed in $O(|\mathcal{A}|^2)$ steps if for example a QR decomposition of the Hessian matrix is maintained and updated at each step [159]. Since $|\mathcal{A}| \leq p$ and typically $k \geq p$, a single event commands $O(kp)$ operations.

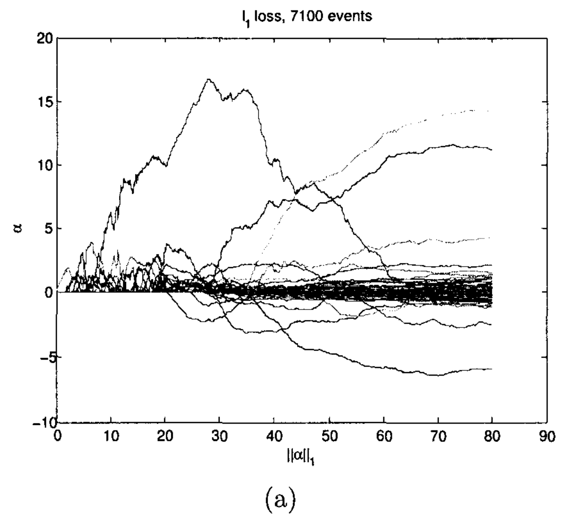
In the case of the l_1 loss function a single event commands the computation of the derivative $\frac{\partial \boldsymbol{v}}{\partial \tau}$ which pessimistically requires the verification of all possibilities of systems (84) and (86). Again, if a QR decomposition of the matrix defining those systems of equations is updated at each stage, this requires $O(p|\mathcal{A}|^2)$ operations. Next, the calculation of $\frac{\partial R}{\partial \tau}$ which is needed to compute the step size commands $O(kp)$ operations, while the selection of the step size requires $O(k+p)$ operations. In total, a single event requires $O(kp+p^3)$ operations because $|\mathcal{A}| \leq p$. The majority of events processed are residues R crossing 0. In consequence, verification of optimality conditions speeds up the algorithm, since usually a solution of (84) is the global optimum and there is no need to check (86).

The coefficients $\boldsymbol{v}$ and residues R may cross zero multiple times making an exact derivation of the number of events processed over a path difficult. When the l_2 loss is used, the only events stem from changes of the active set. When l_H is used one needs also to account for residues crossing knots of the loss function located at $\pm\delta$. Finally, l_1 requires the tracking of every zero-crossing of the residues. This makes it the slowest of the proposed loss functions.

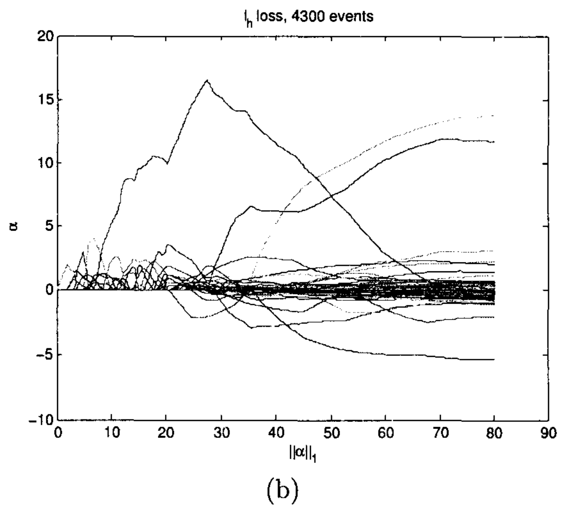
3 Robust Sparse Coding for Face Recognition

To demonstrate the use of robust sparse coding for face recognition (FR) consider a training set of aligned, normalized, and labeled images of faces. Let $\boldsymbol{D}$ be a matrix in which every column $\boldsymbol{D}_{:i}$ contains the pixels of a training face image. Let $\boldsymbol{x}$ be a vector containing the pixels of an unknown face. The recently proposed Sparse Representation-based Classification (SRC) [161] methods first represents $\boldsymbol{x}$ as a sparse linear combination of training faces by solving the problem

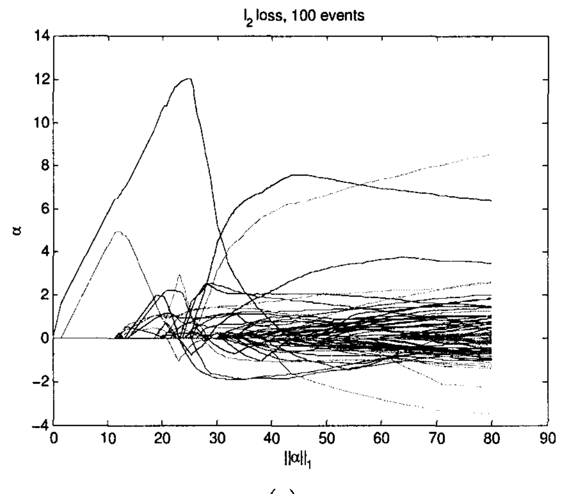
$$\hat{\boldsymbol{v}} = \arg\min_{\boldsymbol{v}} ||\boldsymbol{v}||_1 \text{ subject to: } ||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{v}||_2 \leq \epsilon. \tag{88}$$



(a)



(b)



(c)

Figure 25: Regularization paths for sparse coding coefficients: (a) l_1 error function, (b) l_h error function, (c) l_2 error function. We can see that the Huber loss results are smoother and similar to the l_1 results. The number of "events" processed in each case is indicated. It coincides with the theoretical analysis.

Then per-person residuals r_i of the approximation are computed for every person i in the training data:

$$r_i(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{D}\delta_i(\boldsymbol{v})||_2, \tag{89}$$

where the function $\delta_i(\boldsymbol{x}) : \mathbb{R}^p \to \mathbb{R}^p$ returns a vector whose only nonzero components are the entries of $\boldsymbol{x}$ associated with the person i. The unknown face $\boldsymbol{x}$ is recognized as the person whose residual was the lowest: identity($\boldsymbol{x}$) = $\arg\min_i r_i(\boldsymbol{x})$. The authors of SRC have also proposed an extension for robust recognition [161]. The dictionary of training faces $\boldsymbol{D}$ is extended with an identity matrix serving as a dictionary of noise patterns to form $\boldsymbol{D}_E = [\boldsymbol{D}, \boldsymbol{I}]$ [3]. Then the unknown face $\boldsymbol{x}$ is represented in terms of the training faces and noise jointly:

$$\boldsymbol{x} = [\boldsymbol{D}, \boldsymbol{I}] \begin{bmatrix} \boldsymbol{v} \\ \boldsymbol{v}_E \end{bmatrix}, \tag{90}$$

where $\boldsymbol{v}_E$ are the noise coefficients. Since this system of equations is always under-determined, it is necessary to solve equation (91) to recover the sparsest solution:

$$\begin{gathered} \hat{\boldsymbol{v}}, \hat{\boldsymbol{v}_E} = \underset{\boldsymbol{v}, \boldsymbol{v}_E}{\arg\min} ||\boldsymbol{v}||_1 + ||\boldsymbol{v}_E||_1 \\ \text{subject to: } \boldsymbol{D}\boldsymbol{v} + \boldsymbol{I}\boldsymbol{v}_E = \boldsymbol{x}. \end{gathered} \tag{91}$$

The noise present in the test image $\boldsymbol{x}$ is captured in the coefficients $\boldsymbol{v}_E$. Hence $\boldsymbol{D}\boldsymbol{v} = \boldsymbol{x} - \boldsymbol{I}\boldsymbol{v}_E$ is the reconstructed denoised image. The residuals are redefined to reflect the reconstruction error of denoised images:

$$r_i(\boldsymbol{x}) = ||\boldsymbol{x} - \boldsymbol{I}\boldsymbol{v}_E - \boldsymbol{D}\delta_i(\boldsymbol{v})||_2 = ||\boldsymbol{D}(\boldsymbol{v} - \delta_i(\boldsymbol{v}))||_2. \tag{92}$$

The problem (91) is equivalent to minimizing the ℓ_1 norm of the approximation error with the parameter λ equal to 1:

$$\min ||\boldsymbol{x} - \boldsymbol{D}\boldsymbol{v}||_1 + \lambda ||\boldsymbol{v}||_1. \tag{93}$$

This can easily be seen by interpreting $\boldsymbol{v}_E$ as reconstruction residuals. The parameter λ is implicitly present in (91) as the ratio of magnitudes of elements of $\boldsymbol{D}$ and $\boldsymbol{x}$. In fact the results presented in [161] require that only the training faces are normalized, while test faces are not. The inclusion of the regularization parameter λ in (70) and (93), or the parameter ϵ in (88) makes this dependence explicit.

4 Experimental Results

In all the experiments the Extended Yale B face recognition database was used [168]. Selected were the 719 face images taken with capture angle and elevation

[3]If the noise is known to have a sparse structure with respect to another basis, [161] proposes to use this basis instead of the identity matrix.

lower than 25° as the training set and 373 face images with capture angle and elevation between 25° and 50° as the test set. Similarly to [161] the training faces were normalized to have unit length, while the test faces were unnormalized and corrupted by replacing 30% pixels with random values drawn uniformly from the range of pixel intensity values in the testing set. In all experiments the parameter δ of the Huber loss function was set to 1/10 of the standard deviation of intensity values of test images, which was 0.028.

First compared are the coefficients obtained using the l_1, l_2, and l_h error measures for a single test image. The results are presented in Figure 25. The path obtained during minimization of the l_2 error function has the smallest number of segments (the algorithm processed the fewest events), however it differs greatly from the paths obtained for l_1 and l_h error functions. The path obtained with l_h is smoother than the one obtained for l_1.

Table 13 compares execution times required to solve the problem (71) using the proposed algorithm and Matlab's "linprog" function. All algorithms converged to approximately the same vector of optimal coefficients $\boldsymbol{v}$ which validates the correctness of the proposed algorithm. To assess the speedup due to early termination of search for possible solutions of (81) by checking optimality conditions the time required when optimality checks were disabled is also reported. The results indicate that for large values of λ when the solutions are sparse, the proposed approach is competitive with directly solving (71) using linear programming because the experimental running times are comparable and the full regularization path is obtained instead of a single solution. However, as the solution becomes less sparse it is more beneficial to repeatedly use linear programming techniques for several values of the regularization parameter. The checks of optimality conditions result in a considerable speedup, however the relative gap becomes smaller for less sparse solutions because there are more satisfied (equal to zero) residuals which always need to be considered and less inactive (equal to zero) coefficients.

Finally, the accuracy of face recognition was calculated for a range of regularization parameter values and plotted in Figure 26. For every testing image the

TABLE 13

Running times of the proposed algorithm compared with running times of Matlab's "linprog" solver.

	λ			
	1.0	0.1	0.02	0.01
linprog time [s]	89	126	170	266
with opt. checks [s]	21	200	5050	14700
without opt. checks [s]	370	2100	19400	-
processed events	3900	8000	14400	17600

Relative differences between different algorithm's solutions were $\leq 10^{-5}$.

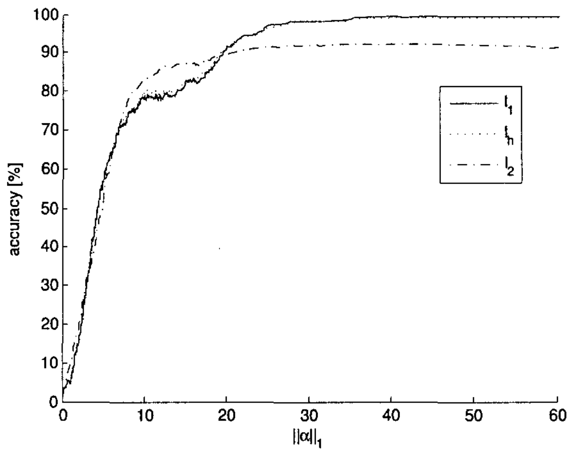


Figure 26: Accuracy of sparse-coding based face recognition on the Extended Yale B database with added noise. The two robust error measures give very close results, with l_1 being slightly better than l_h.

values of $||\boldsymbol{v}||_1$ were determined at which the proper label was beginning or ending to be selected and those values were added to compute how many faces are correctly recognized for every value of $||\boldsymbol{v}||_1$. The error function l_2 is affected by the added noisy pixels and yields the lowest accuracy. The Huber and ℓ_1-norm based error functions are more robust and yield nearly identical results, with the l_1 loss function being slightly better. The obtained results are comparable with the ones reported in [161].

5 Conclusions

A novel algorithm for obtaining the full regularization path of the robust sparse coding problem was presented. Optimality conditions were derived through the use of subgradient calculus. The conditions were successfully incorporated into the algorithm obtaining yielding important running time savings. A reasoning similar to the one presented in this paper can also be used to speed up the L1-SVM [164] method. The proposed algorithm was benchmarked on real-life face recognition data demonstrating its validity and usefulness. It is concluded that when the solution of the robust sparse coding problem is assumed to be very sparse, the proposed approach is competitive with linear programming solvers.

CHAPTER VI

CONCLUSIONS

The task of learning understandable concepts and relations from data is a complex and multifaceted one. The notion of understandability is subjective and can only be measured indirectly, usually by the expressiveness and size of representations that describe concepts and patterns. Therefore, there is no unique way of enforcing the understandability of results of a given method. However, then main conclusions that arise from the body of research reported upon in this dissertation is that understandability requires both that the induced descriptions be interpretable and relevant to the problem at hand.

The importance of concentrating on the relevant characteristics of the problem is demonstrated in Chapter III, in which it was observed that methods which generated additional data in the regions relevant to the problem have led to the smallest and most accurate sets of rules. This suggests that for rule extraction it is neither useful nor correct to attempt to describe all the details of the black-box's operation. Instead, only the patterns relevant to the problem should be analyzed.

Two rule extraction methods proposed in Chapter IV are specialized algorithms that closely rely on properties of both the black-box classifier they attempt to understand – an Artificial Neural Network, and the properties of the understandable data structure they derive – a Reduced Ordered Decision Diagram. However, they incorporate the principle of extracting only the relevant patterns as they both limit the search space to the proximity of the training data.

Finally, the neural network with non-negative weights proposed in Chapter V was designed to detect patterns that are both understandable and relevant to the classification task defined by the target labeling. This is in sharp contrast to traditional feature detection methods that try to faithfully describe the data and disregard the ultimate task of classification.

Methods described in this dissertation encompass several key stages of data analysis process: understandable feature discovery, efficient data encoding using known features, and finally induction of understandable models. It is hoped that the presented algorithms will extend the toolkit of data analysts and enable the extraction of useful knowledge out of data. The original contributions of the author of this dissertation include:

- The introduction of a definition of understandability that can be used in formal proofs of the computational complexity of rule learning.
- The extension of kernel density estimation methods to support nominal multivalued attributes which is needed in practical applications.
- The introduction of two novel algorithms that use the ability to estimate the activation of a Neural Network in the presence of unknown inputs to induce Reduced Ordered Decision Diagrams.
- The introduction of the non-negative weight constraints in Neural Networks to enhance their understandability and a proof of the ability of a sufficiently large network with non-negative weight constraints to shatter a given set of points.
- The introduction of a novel path-following algorithm for the robust sparse coding problem.

REFERENCES

[1] R. Michalski and R. Chilausky, "Knowledge acquisition by encoding expert rules versus computer induction from examples: a case study involving soybean pathology," *Int. J. of Man-Machine Studies*, vol. 12, pp. 63–87, 1980.

[2] W. Duch, R. Setiono, and J. Zurada, "Computational intelligence methods for rule-based data understanding," *Proceedings of the IEEE*, vol. 92, no. 5, pp. 771–805, May 2004.

[3] M. Nguyen, J. Zurada, and J. Rajapakse, "Toward Better Understanding of Protein Secondary Structure: Extracting Prediction Rules," *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, vol. 8, no. 3, pp. 858–864, Jun. 2011.

[4] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," in *Proc. 23rd Int. Conf. on Mach. Learning*, ser. ICML '06. New York, NY, USA: ACM, 2006, pp. 161–168.

[5] Y. Freund and R. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learning*, 1996.

[6] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, pp. 123–140, 1996.

[7] ——, "Random forests," *Mach. Learn.*, vol. 45, pp. 5–32, 2001.

[8] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[9] J. M. Zurada, *Introduction to artificial neural systems.* West Publishing Company, 1992.

[10] V. Cherkassky and F. Mulier, *Learning from data: concepts, theory, and methods.* Wiley-IEEE Press, 2007.

[11] V. Vapnik, *Statistical learning theory.* Wiley, New York, 1998.

[12] V. N. Vapnik, "An overview of statistical learning theory." *Neural Networks, IEEE Transactions on*, vol. 10, no. 5, pp. 988–99, Jan. 1999.

[13] J. R. Quinlan, *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[14] W. W. Cohen, "Fast effective rule induction," in *Proc. 12th Int. Conf. Mach. Learning*, A. Prieditis and S. Russell, Eds., vol. 3. Morgan Kaufmann, 1995, pp. 115–123.

[15] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[16] F. Supek. (2009, March) Fast random forest v. 0.98. Accessed 9/14/2012. [Online]. Available: http://code.google.com/p/fast-random-forest/

[17] A. Frank and A. Asuncion, "UCI machine learning repository," 2012. [Online]. Available: http://archive.ics.uci.edu/ml

[18] L. Valiant, "A theory of the learnable," *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1142, 1984.

[19] A. Blumer and A. Ehrenfeucht, "Learnability and the Vapnik-Chervonenkis dimension," *Journal of the ACM*, vol. 36, no. 4, pp. 929–965, 1989.

[20] Y. LeCun, L. Bottou, G. Orr, and K. Müller, "Efficient backprop," *Neural networks: Tricks of the trade*, vol. 1524, no. 3, 1998.

[21] M. Mø ller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks*, vol. 6, no. 4, pp. 525–533, 1993.

[22] M. Hagan and M. Menhaj, "Training feedforward networks with the Marquardt algorithm," *Neural Networks, IEEE Transactions on*, vol. 5, no. 6, pp. 989–993, 1994.

[23] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks," *Knowledge-Based Systems*, vol. 8, no. 6, pp. 373–389, Dec. 1995.

[24] R. Reed, "Pruning algorithms-a survey," *Neural Networks, IEEE Transactions on*, vol. 4, no. 5, pp. 740–747, 1993.

[25] Y. Le Cun, J. Denker, S. Solla, R. Howard, and L. Jackel, "Optimal brain damage," *Advances in neural information processing systems*, vol. 2, 1990.

[26] B. Hassibi, D. Stork, and G. Wolff, "Optimal brain surgeon and general network pruning," in *Neural Networks, 1993., IEEE International Conference on.* IEEE, 1993, pp. 293–299.

[27] G. Castellano, A. Fanelli, and M. Pelillo, "An iterative pruning algorithm for feedforward neural networks," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 519–31, Jan. 1997.

[28] M. Ishikawa, "Structural learning with forgetting," *Neural Networks*, vol. 9, no. 3, pp. 509–521, Apr. 1996.

[29] 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, Apr. 2005.

[30] S. Nowlan and G. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural computation*, vol. 4, no. 4, pp. 473–493, 1992.

[31] R. Setiono, "Extracting M-of-N rules from trained neural networks," *Neural Networks, IEEE Transactions on*, vol. 11, no. 2, pp. 512–519, 2000.

[32] B. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*, 1992, pp. 144–152.

[33] C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[34] N. Cristianini and J. Shawe-Taylor, *An introduction to support Vector Machines: and other kernel-based learning methods.* Cambridge Univ Pr, 2000.

[35] P. Tan, M. Steinbach, V. Kumar *et al.*, *Introduction to data mining.* Pearson Addison Wesley Boston, 2006.

[36] C. Bishop, *Pattern recognition and machine learning.* springer New York, 2006.

[37] J. Nocedal and S. Wright, *Numerical optimization.* Springer verlag, 1999.

[38] J. Platt, "Fast Training of Support Vector Machines using Sequential Minimal Optimization," in *Advances in kernel methods*, B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. MIT Press, 1998, pp. 185–208.

[39] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 140, pp. 123–140, 1996.

[40] T. K. Ho, "The random subspace method for constructing decision forests," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 8, pp. 832–844, 1998.

[41] R. E. Schapire, Y. Freund, P. Bartlett, and W. S. Lee, "Boosting the margin: a new explanation for the effectiveness of voting methods," *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686, Oct. 1998.

[42] S. Rosset, "Boosting as a Regularized Path to a Maximum Margin Classifier," *The Journal of Machine Learning Research*, vol. 5, pp. 941–973, 2004.

[43] B. Baesens, R. Setiono, C. Mues, and J. Vanthienen, "Using Neural Network Rule Extraction and Decision Tables for Credit-Risk Evaluation," *Management Science*, vol. 49, no. 3, pp. 312–329, Mar. 2003.

[44] B. Gaines, "Transforming rules and trees into comprehensible knowledge structures," in *Advances in Knowledge Discovery and Data Mining.* MIT Press, 1996, pp. 205–226.

[45] P. Clark and T. Niblett, "The cn2 induction algorithm," *Mach. Learn.*, vol. 3, pp. 261–283, Mar. 1989.

[46] E. Frank and I. Witten, "Generating accurate rule sets without global optimization," in *Proc.of the 15th Int. Conf. on Mach. Learn.*, 1998, pp. 144–151.

[47] J. Furnkranz and P. Flach, "Roc 'n' rule learning–towards a better understanding of covering algorithms," *Mach. Learning*, vol. 58, pp. 39–77, 2005.

[48] L. Breiman, J. Friedman, C. Stone, and R. Olshen, *Classification and Regression Trees*, 1st ed. Chapman & Hall/CRC, Jan. 1984.

[49] J. R. Quinlan, "Learning logical definitions from relations," *Machine Learning*, vol. 5, pp. 239–266, 1990.

[50] M. Kantardzic, *Data Mining: Concepts, Models, Methods and Algorithms.* New York, NY, USA: John Wiley & Sons, Inc., 2002.

[51] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F.-J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, G. Bakir, T. Hofman, B. Schölkopf, A. Smola, and B. Taskar, Eds. MIT Press, 2006.

[52] M. A. Ranzato, Y.-L. Boureau, S. Chopra, and Y. LeCun, "A unified energy-based framework for unsupervised learning," in *AISTATS*, 2007.

[53] M. Ranzato, "Unsupervised Learning of Feature Hierarchies," Ph.D. dissertation, New York University, 2009.

[54] G. Golub and C. Van Loan, *Matrix computations.* Johns Hopkins University Press, 1996, vol. 3.

[55] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications." *Neural Networks*, vol. 13, no. 4-5, pp. 411–30, 2000.

[56] P. Paatero, "Least squares formulation of robust non-negative factor analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 37, no. 1, pp. 23–35, May 1997.

[57] D. D. Lee and H. S. Seung, "Learning the parts of objects by non-negative matrix factorization." *Nature*, vol. 401, no. 6755, pp. 788–91, Oct. 1999.

[58] 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.

[59] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area V2," in *Advances in neural information processing systems*, 2007, pp. 873–880.

[60] M. Y. Ranzato, L. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," in *Advances in neural information processing systems*, 2007, pp. 1185–1192.

[61] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09*, 2009, pp. 609–616.

[62] J. Huysmans, B. Baesens, and J. Vanthienen, "Using Rule Extraction to Improve the Comprehensibility of Predictive Models," K.U. Leuven, Tech. Rep., 2006. [Online]. Available: http://www.ssrn.com/abstract=961358

[63] I. Cloete and J. Zurada, Eds., *Knowledge-based neurocomputing.* Cambridge, MA, USA: MIT Press, 2000.

[64] Z.-H. Zhou and Y. Jiang, "Medical Diagnosis with C4. 5 Rule Preceded by Artificial Neural Network Ensemble," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 7, no. 1, pp. 37–42, Mar. 2003.

[65] D. Martens, B. Baesens, and T. Van Gestel, "Decompositional Rule Extraction from Support Vector Machines by Active Learning," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 21, no. 2, pp. 178–191, Feb. 2009.

[66] M. Craven and J. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," *Advances in Neural Information Processing Systems*, vol. 8, 1996.

[67] J. Huysmans, Johan and Baesens, Bart and Vanthienen, "ITER: an algorithm for predictive regression rule extraction," in *Data Warehousing and Knowledge Discovery*, vol. 4081, 2006, pp. 270–279.

[68] J. Huysmans, R. Setiono, B. Baesens, and J. Vanthienen, "Minerva: sequential covering for rule extraction." *Systems, Man, and Cybernetics Part B, IEEE Transactions on*, vol. 38, no. 2, pp. 299–309, Apr. 2008.

[69] O. Boz, "Extracting decision trees from trained neural networks," in *Proc. of the 8th ACM SIGKDD Int. Conf.* New York, New York, USA: ACM Press, 2002, pp. 456–461.

[70] T. Etchells and P. Lisboa, "Orthogonal search-based rule extraction (OSRE) for trained neural networks: a practical and efficient approach," *Neural Networks, IEEE Transactions on*, vol. 17, no. 2, pp. 374–384, 2006.

[71] J. R. Rabuñal, J. Dorado, A. Pazos, J. Pereira, and D. Rivero, "A New Approach to the Extraction of ANN Rules and to Their Generalization Capacity Through GP," *Neural computation*, vol. 16, pp. 1483–523, Jul. 2004.

[72] G. J. Schmitz, C. Aldrich, and F. S. Gouws, "ANN-DT: An Algorithm for Extraction of Decision Trees from Artificial Neural Networks," *Neural Networks, IEEE Transactions on*, vol. 10, no. 6, pp. 1392–401, Jan. 1999.

[73] M. Craven and J. Shavlik, "Using sampling and queries to extract rules from trained neural networks," in *Proc. 11th Int. Conf. Mach. Learning*, 1994.

[74] R. Krishnan, G. Sivakumar, and P. Bhattacharya, "Extracting decision trees from trained neural networks," *Pattern Recognition*, vol. 32, no. 12, pp. 1999–2009, Dec. 1999.

[75] D. Specht, "Probabilistic Neural Networks," *Neural networks*, vol. 3, 1990.

[76] E. Saad and D. Wunsch, "Neural network explanation using inversion," *Neural Networks*, vol. 20, no. 1, pp. 78–93, Jan. 2007.

[77] N. Barakat and J. Diederich, "Eclectic Rule-Extraction from Support Vector Machines," *International Journal of Computational Intelligence*, vol. 2, no. 1, pp. 59–62, 2005.

[78] P. Domingos, "Knowledge Acquisition from Examples via Multiple Models," in *Proc. 14th Int. Conf. Mach. Learning*, 1997, pp. 98–106.

[79] R. Setiono, B. Baesens, and C. Mues, "Recursive neural network rule extraction for data with mixed attributes," *Neural Networks, IEEE Transactions on*, vol. 19, no. 2, pp. 299 -307, Feb. 2008.

[80] R. Setiono and W. Leow, "Fernn: An algorithm for fast extraction of rules from neural networks," *Applied Intelligence*, vol. 12, pp. 15–25, 2000.

[81] A. Van Assche and H. Blockeel, "Seeing the forest through the trees: Learning a comprehensible model from an ensemble," in *Machine Learning: ECML 2007*, ser. Lecture Notes in Computer Science, J. Kok, J. Koronacki, R. Mantaras, S. Matwin, D. Mladenic, and A. Skowron, Eds. Springer Berlin / Heidelberg, 2007, vol. 4701, pp. 418–429.

[82] L. Fu, "Rule generation from neural networks," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 24, no. 8, pp. 1114 –1124, Aug. 1994.

[83] G. Towell and J. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine Learning*, vol. 13, pp. 71–101, 1993.

[84] M. Craven and J. Shavlik, "Rule extraction: Where do we go from here," University of Wisconsin Machine Learning Research Group, Tech. Rep., 1999.

[85] I. Taha and J. Ghosh, "Symbolic Interpretation of Artificial Neural Networks," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 11, no. 3, pp. 448–463, 1999.

[86] J. Chorowski and J. M. Zurada, "Extracting Rules from Neural Networks as Decision Diagrams," *Neural Networks, IEEE Transactions on*, vol. 22, no. 12, pp. 2435–46, Dec. 2011.

[87] J. Zurada, A. Malinowski, and S. Usui, "Perturbation method for deleting redundant inputs of perceptron networks," *Neurocomputing*, vol. 14, no. 96, pp. 177–193, 1997.

[88] L. Breiman and A. Cutler, "Random forest – classification description," jun 2004, accessed 9/25/2012. [Online]. Available: http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

[89] J. Vanthienen, "Using Rule Extraction in Data Mining Models: Some New Measures," in *Proceedings of the XV international conference Knowledge Acquisition and Management*, Wrocław, Poland, 2007.

[90] R. S. Michalski, "Attributional Calculus A logic and Representation Language for Natural Induction," Reports of tthe Machine Learning and Inference Laboratory, George Mason University, Tech. Rep., 2004.

[91] G. Huang, Q. Zhu, and C. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[92] J. Wang, J. Chorowski, and J. M. Zurada, "Review and performance comparison of svm- and elm- based classifiers," *submitted to Neurocomputing*, 2012.

[93] G. Towell and J. Shavlik, "Extracting refined rules from knowledge-based neural networks," *Machine learning*, vol. 13, pp. 71–101, 1993.

[94] J. Proakis and D. Manolakis, *Digital Signal Processing*, 4th ed. Prentice Hall, 2006.

[95] D. Lowe, "Object recognition from local scale-invariant features," in *ICCV*. Published by the IEEE Computer Society, 1999, p. 1150.

[96] T. Ojala, M. Pietikäinen, and D. Harwood, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[97] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[98] J. Wang, Z. Liu, J. Chorowski, Z. Chen, and Y. Wu, "Robust 3D Action Recognition with Random Occupancy Patterns," in *ECCV 2012*, 2012, pp. 872–885.

[99] D. Hubel and T. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[100] M. Y. Ranzato, P. Christopher, C. Sumit, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," *Advances in Neural Information Processing Systems*, 2006.

[101] H. Lee, R. Grosse, R. Ranganath, and A. Ng, "Unsupervised learning of hierarchical representations with convolutional deep belief networks," *Communications of the ACM*, vol. 54, no. 10, pp. 95–103, 2011.

[102] Y. S. Abu-Mostafa, "Learning from hints in neural networks," *Journal of Complexity*, vol. 6, no. 2, pp. 192–198, Jun. 1990.

[103] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality Reduction by Learning an Invariant Mapping," *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, pp. 1735–1742, 2006.

[104] J. Chorowski and J. M. Zurada, "Improving the accuracy of understandable classifiers through additional sample generation," *Submitted to Knowledge and Data Engineering, IEEE Transactions on*, 2012.

[105] N. Chawla, K. Bowyer, L. Hall, and W. Kegelmeyer, "SMOTE: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[106] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in large margin classifiers*, A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, Eds. MIT Press, 1999.

[107] P. Sollich, "Probabilistic methods for support vector machines," *Advances in neural information processing systems*, pp. 349–355, 2000.

[108] B. Silverman, *Density estimation for statistics and data analysis.* Chapman and Hall, 1986.

[109] D. Scott, *Multivariate density estimation.* John Wiley and Sons, 1992, vol. 139.

[110] J. Hwang, S. Lay, and A. Lippman, "Nonparametric multivariate density estimation: a comparative study," *Signal Processing, IEEE Transactions on*, vol. 42, no. 10, pp. 2795–2810, 1994.

[111] B. Park and B. Turlach, "Practical performance of several data driven bandwidth selectors," *Computational Statistics*, vol. 7, pp. 251–285, 1992.

[112] M. Jones, J. Marron, and S. Sheather, "A brief survey of bandwidth selection for density estimation," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 401–407, 1996.

[113] A. Hyvärinen, "Fast and Robust Fixed-Point Algorithms for Independent Component Analysis." *Neural Networks, IEEE Transactions on*, vol. 10, no. 3, pp. 626–34, Jan. 1999.

[114] 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, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[115] R. Bryant, "Graph-based algorithms for boolean function manipulation," *Computers, IEEE Transactions on*, vol. C-35, no. 8, pp. 677 –691, Aug. 1986.

[116] Y.-T. Lai and S. Sastry, "Edge-valued binary decision diagrams for multi-level hierarchical verification," in *Proceedings of the 29th ACM/IEEE Design Automation Conference*, ser. DAC '92. Los Alamitos, CA, USA: IEEE Computer Society Press, 1992, pp. 608–613.

[117] P. Tafertshofer and M. Pedram, "Factored edge-valued binary decision diagrams," *Formal Methods in System Design*, vol. 10, pp. 243–270, 1997.

[118] J. Ossowski and C. Baier, "A uniform framework for weighted decision diagrams and its implementation," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 10, pp. 425–441, 2008.

[119] R. Bryant and Y.-A. Chen, "Verification of arithmetic circuits using binary moment diagrams," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 3, pp. 137–155, 2001.

[120] R. Bryant, "Binary decision diagrams and beyond: enabling technologies for formal verification," in *Computer-Aided Design, 1995. ICCAD-95. Digest of Technical Papers., 1995 IEEE/ACM International Conference on*, Nov. 1995, pp. 236 –243.

[121] C. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design*, 1st ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1998.

[122] C. Files and M. Perkowski, "Multi-valued functional decomposition as a machine learning method," in *Multiple-Valued Logic, 1998. Proceedings. 1998 28th IEEE International Symposium on*, May 1998, pp. 173 –178.

[123] A. Oliveira and A. Sangiovanni-Vincentelli, "Learning complex boolean functions: Algorithms and applications," in *In Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1993, pp. 911–918.

[124] M. Sauerhoff and I. Wegener, "On the complexity of minimizing the obdd size for incompletely specified functions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 11, pp. 1435 –1437, Nov. 1996.

[125] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. Brayton, "Heuristic minimization of bdds using don't cares," in *Proceedings of the 31st annual Design Automation Conference*, ser. DAC '94. New York, NY, USA: ACM, 1994, pp. 225–231.

[126] Y. Hong, P. Beerel, J. Burch, and K. McMillan, "Safe bdd minimization using don't cares," in *Proceedings of the 34th annual Design Automation Conference*, ser. DAC '97. New York, NY, USA: ACM, 1997, pp. 208–213.

[127] A. Oliveira, L. Carloni, T. Villa, and A. Sangiovanni-Vincentelli, "Exact minimization of binary decision diagrams using implicit techniques," *Computers, IEEE Transactions on*, vol. 47, no. 11, pp. 1282 –1296, Nov. 1998.

[128] R. Kohavi, "Bottom-up induction of oblivious read-once decision graphs: strengths and limitations," in *Proceedings of the 12th national conference on Artificial intelligence (vol. 1)*, ser. AAAI '94. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1994, pp. 613–618.

[129] A. Oliveira and A. Sangiovanni-Vincentelli, "Using the minimum description length principle to infer reduced ordered decision graphs," *Machine Learning*, vol. 25, pp. 23–50, 1996.

[130] C. Mues, B. Baesens, C. Files, and J. Vanthienen, "Decision diagrams in machine learning: an empirical study on real-life credit-risk data," *Expert Systems with Applications*, vol. 27, no. 2, pp. 257 – 264, 2004.

[131] K. Brace, R. Rudell, and R. Bryant, "Efficient implementation of a bdd package," in *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, Jun. 1990, pp. 40 –45.

[132] H. Andersen, "An introduction to binary decision diagrams," IT University of Copenhagen, Lect. Not., 1999. [Online]. Available: http://www.itu.dk/people/hra/notes-index.html

[133] S. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. D. Jong, S. Dzeroski, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. Michalski, T. Mitchell, P. Pachowicz, B. Roger, H. Vafaie, W. V. de Velde, W. Wenzel, J. Wnek, and J. Zhang, "The MONK's problems: A performance comparison of different learning algorithms," Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, Tech. Rep. CMU-CS-91-197, 1991.

[134] S. Tani, K. Hamaguchi, and S. Yajima, "The complexity of the optimal variable ordering problems of shared binary decision diagrams," in *Algorithms and Computation*, ser. Lecture Notes in Computer Science, K. Ng, P. Raghavan, N. Balasubramanian, and F. Chin, Eds. Springer Berlin / Heidelberg, 1993, vol. 762, pp. 389–398.

[135] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD '93. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993, pp. 42–47.

[136] S. Panda, F. Somenzi, and B. Plessier, “Symmetry detection and dynamic variable ordering of decision diagrams,” in *Proceedings of the 1994 IEEE/ACM international conference on Computer-aided design*, ser. ICCAD ’94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 628–631.

[137] F. Somenzi, “Efficient manipulation of decision diagrams,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 3, pp. 171–181, 2001.

[138] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, “Algebraic decision diagrams and their applications,” *Formal Methods in System Design*, vol. 10, pp. 171–206, 1997.

[139] S. Sanner and D. McAllester, “Affine algebraic decision diagrams (aadds) and their application to structured probabilistic inference,” in *Proceedings of the 19th international joint conference on Artificial intelligence.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, pp. 1384–1390.

[140] J. Bern, J. Gergov, C. Meinel, and A. Slobodova, “Boolean manipulation with free bdd’s. first experimental results,” in *European Design and Test Conference, 1994. EDAC, The European Conference on Design Automation. ETC European Test Conference. EUROASIC, The European Event in ASIC Design, Proceedings.*, 1994, pp. 200 –207.

[141] J. Bern, C. Meinel, and A. Slobodova, “Some heuristics for generating tree-like fbdd types,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 15, no. 1, pp. 127 –130, Jan. 1996.

[142] J. Møller, J. Lichtenberg, H. Andersen, and H. Hulgaard, “Difference decision diagrams,” in *Proceedings of the 13th International Workshop and 8th Annual Conference of the EACSL on Computer Science Logic*, ser. CSL ’99. London, UK: Springer-Verlag, 1999, pp. 111–125.

[143] G. Towell, J. Shavlik, and M. Noordewier, “Refinement of approximate domain theories by knowledge-based neural networks,” in *Proceedings of the Eighth National Conference on Artificial Intelligence.* Citeseer, 1990, pp. 861–866.

[144] J. Chorowski and J. Zurada, “Top-down induction of reduced ordered decision diagrams from neural networks,” *Lecture Notes in Computer Science, Artificial Neural Networks and Machine Learning ICANN 2011*, vol. 6729, pp. 309–316, 2011.

[145] R. Kohavi and C. H. Li, “Oblivious decision trees graphs and top down pruning,” in *Proceedings of the 14th international joint conference on Artificial*

intelligence-Volume 2. Morgan Kaufmann Publishers Inc., 1995, pp. 1071–1077.

[146] U. Fayyad and K. Irani, "Multi-interval discretization of continuous-valued attributes for classification learning," in *Proceedings of the International Joint Conference on Uncertainty in AI*, 1993, pp. 1022–1027.

[147] J. Chorowski and J. M. Zurada, "Learning understandable neural networks with non-negative weight constraints," *Submitted to Neural Networks and Learning Systems, IEEE Transactions on*, 2012.

[148] K. Huang and S. Aviyente, "Sparse Representation for Signal Classification," in *Advances in Neural Information Processing Systems 19*, 2007, pp. 609–616.

[149] J. Mairal, F. Bach, J. Ponce, G. Sapiro, and A. Zisserman, "Discriminative learned dictionaries for local image analysis," in *IEEE Conference on Computer Vision and Pattern Recognition, 2008. CVPR 2008*, Jun. 2008.

[150] J. Mairal, F. Bach, A. Zisserman, G. Sapiro, J. Ponce, G. Sapiro, and A. Zisserman, "Supervised dictionary learning," in *Advances in Neural Information Processing Systems 21*, 2009, pp. 1033–1040.

[151] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[152] M. Y. Ranzato and M. Szummer, "Semi-supervised learning of compact document representations with deep networks," in *International Conferenece on Machine Learning, IMCL 25*, 2008.

[153] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software*, vol. 23, no. 4, pp. 550–560, Dec. 1997.

[154] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of Machine Learning Research*, vol. 5, pp. 1457–1469, Aug. 2004.

[155] J. Chorowski and J. Zurada, "Obtaining full regularization paths for robust sparse coding with applications to face recognition." in *Accepted to International Conference on Machine Learing and Applications - ICMLA*, 2012.

[156] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[157] M. R. Osborne, B. Presnell, and B. A. Turlach, "On the lasso and its dual," *Journal of Computational and Graphical Statistics*, vol. 9, no. 2, pp. 319–337, 2000.

[158] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[159] S. Rosset and J. Zhu, "Piecewise linear regularized solution paths," *The Annals of Statistics*, vol. 35, no. 3, pp. 1012–1030, Jul. 2007.

[160] M. Park and T. Hastie, "L1regularization path algorithm for generalized linear models," *Journal of the Royal Statistical Society:*, vol. 69, no. 4, pp. 659–677, 2007.

[161] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma, "Robust face recognition via sparse representation." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 2, pp. 210–227, 2009.

[162] L. Qiao, S. Chen, and X. Tan, "Sparsity preserving projections with applications to face recognition," *Pattern Recognition*, vol. 43, no. 1, pp. 331–341, Jan. 2010.

[163] J. Gui, Z. Sun, W. Jia, R. Hu, Y. Lei, and S. Ji, "Discriminant sparse neighborhood preserving embedding for face recognition," *Pattern Recognition*, vol. 45, no. 8, pp. 2884–2893, 2012.

[164] J. Zhu, S. Rosset, and T. Hastie, "1-norm support vector machines," in *Advances in Neural Information Processing Systems*, 2004, pp. 49–56.

[165] Y. Nesterov, *Introductory Lectures On Convex Optimization: A Basic Course.* Kluwer Academic Publishers, 2004.

[166] S. Boyd, "EE3642: Lecture Slides and Notes," 2011. [Online]. Available: http://www.stanford.edu/class/ee364b/lectures.html

[167] R. T. Rockafellar, *Convex Analysis.* Princeton University Press, 1970.

[168] K.-C. Lee, J. Ho, and D. J. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 5, pp. 684–98, May 2005.

CURRICULUM VITAE

NAME:	Jan Chorowski
ADDRESS:	316 W. Lee St., Apt. 3 Louisville, KY 40208
EDUCATION & TRAINING:	M.Eng., Microsystem Electronics and Photonics Wroclaw University of Technology 2004 - 2009
	Mathematics and Computer Science University of Wroclaw 2005 - 2009
	Ph.D., Electrical and Computer Engineering University of Louisville 2009 - 2012
	Microsoft Research Summer Internship Microsoft Research Redmond 2011 (12 weeks)
TEACHING:	Electronics I Lab- GTA Communication Systems Lab - GTA Control Systems Principles Lab - GTA
AWARDS:	Electrical Engineering Outstanding Graduate Student Award, University of Louisville, 2011
	Best Graduate Student Award Wroclaw University of Technology, 2009
PROFESSIONAL SOCIETIES:	Institute of Electrical and Electronic Engineers

PUBLICATIONS

REFEREED JOURNALS

J. Chorowski and J. M. Zurada, "Improving the accuracy of understandable classifiers through additional sample generation," *In Preparation for Knowledge and Data Engineering, IEEE Transactions on.*

J. Chorowski and J. M. Zurada, "Learning understandable neural networks with non-negative weight constraints," *Submitted to Neural Networks and Learning Systems, IEEE Transactions on*, 2012.

J. Wang, J. Chorowski, and J. M. Zurada, "Review and performance comparison of svm- and elm- based classifiers," *submitted to Neurocomputing*, 2012.

J. Chorowski and J. M. Zurada, "Extracting Rules from Neural Networks as Decision Diagrams," *Neural Networks, IEEE Transactions on*, vol. 22, no. 12, pp. 2435–46, Dec. 2011.

CONFERENCES

J. Chorowski and J. Zurada, "Obtaining full regularization paths for robust sparse coding with applications to face recognition." *Accepted to International Conference on Machine Learing and Applications - ICMLA 2012.*

T. Ensari, J. Chorowski, J.M. Zurada, "Occluded Face Recognition Using Correntropy-based Nonnegative Matrix Factorization", *Accepted to International Conference on Machine Learning and Applications - ICMLA 2012.*

J. Wang, Z. Liu, J. Chorowski, Z. Chen, and Y. Wu, "Robust 3D Action Recognition with Random Occupancy Patterns," in *Computer Vision - ECCV 2012*, pp. 872–885.

T. Ensari, J. Chorowski, J.M. Zurada, "Correntropy-based Document Clustering via Nonnegative Matrix Factorization", in *Lecture Notes in Computer Science, Proc. of the 22nd International Conference on Artificial Neural Networks - ICANN 2012*, vol. 7553, pp. 347-354.

J. Chorowski and J. Zurada, "Top-down induction of reduced ordered decision diagrams from neural networks," *Lecture Notes in Computer Science, Artificial Neural Networks and Machine Learning - ICANN 2011*, vol. 6729, pp. 309–316, 2011.

J. Chorowski, T. Wiśniowski, M. Czok, D. Jurków, K. Malecha, K. Chmiel-Kurowska, L. Golonka, "LTCC module for the investigation of thermal properties of nanofluids", in *Proc. of the XXXIII International Conference of IMAPS - CPMT IEEE* Poland, Pszczyna, September 21-24th 2009, pp. 31.