Abstract

Title of dissertation: SCALABLE LEARNING
FOR GEOSTATISTICS AND
SPEAKER RECOGNITION

Balaji Vasan Srinivasan
Doctor of Philosophy, 2011

Thesis directed by: Professor Ramani Duraiswami
Department of Computer Science

With improved data acquisition methods, the amount of data that is being collected has increased several fold. One of the objectives in data collection is to learn useful underlying patterns. In order to work with data at this scale, the methods not only need to be effective with the underlying data, but also have to be scalable to handle larger data collections. This thesis focuses on developing scalable and effective methods targeted towards different domains, geostatistics and speaker recognition in particular.

Initially we focus on kernel based learning methods and develop a GPU based parallel framework for this class of problems. An improved numerical algorithm that utilizes the GPU parallelization to further enhance the computational performance of kernel regression is proposed. These methods are then demonstrated on problems arising in geostatistics and speaker recognition.

In geostatistics, data is often collected at scattered locations and factors like

instrument malfunctioning lead to missing observations. Applications often require the ability to interpolate this scattered spatiotemporal data on to a regular grid continuously over time. This problem can be formulated as a regression problem, and one of the most popular geostatistical interpolation techniques, kriging is analogous to a standard kernel method: Gaussian process regression. Kriging is computationally expensive and needs major modifications and accelerations in order to be used practically. The GPU framework developed for kernel methods is extended to kriging and further the GPU's texture memory is better utilized for enhanced computational performance.

Speaker recognition deals with the task of verifying a person's identity based on samples of his/her speech utterances. This thesis focuses on text-independent framework and three new recognition frameworks were developed for this problem. We proposed a kernelized Renyi distance based similarity scoring for speaker recognition. While its performance is promising, it does not generalize well for limited training data and therefore does not compare well to state-of-the-art recognition systems. These systems compensate for the variability in the speech data due to the message, channel variability, noise and reverberation. State-of-the-art systems model each speaker as a mixture of Gaussians (GMM) and compensate for the variability (termed nuisance). We propose a novel discriminative framework using a latent variable technique, partial least squares (PLS), for improved recognition. The kernelized version of this algorithm is used to achieve a state-of-the-art speaker ID system, that shows results competitive with the best systems reported on in NISTs 2010 Speaker Recognition Evaluation.

SCALABLE LEARNING FOR GEOSTATISTICS AND SPEAKER RECOGNITION

by

Balaji Vasan Srinivasan

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:
Professor Ramani Duraiswami, Chair/Advisor
Professor Shihab Shamma, Dean's Representative
Professor Larry Davis
Professor Rama Chellappa
Professor Howard Elman

Dedication

I dedicate this thesis to my parents who have been supporting me in all my endeavors and are instrumental in what I am in this life of mine. I also place all my work as an offering to the Lotus feet of Bhagawan Sri Sathya Sai Baba.

Acknowledgments

I am grateful to those people who have made this research and thesis possible and those who have made my experience in graduate school one that I will always remember fondly.

First and foremost I'd like to thank my advisor, Professor Ramani Duraiswami for giving me an invaluable opportunity to work on challenging and extremely interesting projects. I should say I have been amazingly fortunate to have had an advisor who has always been a perfect friend, philosopher and guide in all walks of my graduate life. His unparalleled enthusiasm and energy-level have inspired me during tough times. He gave me the freedom to explore on my own, and at the same time the guidance to recover when my steps faltered. I am always thankful to him to have made my PhD such a beautiful learning experience!

I thank Professors Larry Davis, Rama Chellappa, Shihab Shamma and Howard Elman for agreeing to serve on my dissertation committee. I have had the wonderful opportunity to interact with each of them at varying degrees during different times of my dissertation research. I have learned a lot through these interactions as well as the courses I have taken with them. They have all been instrumental in shaping up different parts of my research. I would also like to thank Professor Raghu Murtugudde for his invaluable inputs in my work on kriging.

I thank Drs. Dmitry Zotkin and Nail Gumerov for all the discussions and brainstorming sessions I have had with them during the course of my research. I would also like to thank my colleagues Adam O'donovan, Qi Hu and Yuancheng

(Mike) Luo for the constructive criticisms and discussions that have helped me improve upon my work.

I would also like to gratefully acknowledge the National Ocean and Atmospheric Administration (NOAA) [Award NA06NES4280016] and the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the Army Research Laboratory (ARL) for funding various parts of the research in this thesis.

A special thanks to Bargava, Raghu, Ashwin and Kiran for providing useful guidance at crucial moments. I also thank my friends Ramanand, Ranjit, Sidharth, Suhasini, Poornima, Karthik and Annamalai for all the bright moments that I have shared with them during my PhD days.

I am forever indebted to my parents because I am what I am due to their support, effort and sacrifices. They have been by my side at all situation giving me the much needed emotional support always.

Finally, I thank Almighty for giving me the physical and mental strength to work on this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

During the past decade, it has become relatively easy to collect huge amounts of data. Examples include data in astronomy, internet traffic, meteorology and surveillance. A goal of this collection is to mine the data for useful information and thus learn meaningful statistical patterns that allow one to predict/recognize unseen patterns.

1.1 Learning methods

Learning is a principled method for distilling predictive and scientific theories from raw data. There are different flavors to learning. In *regression*, a few observations are used to model a continuous target variable, e.g. predicting the temperature/rainfall at some point in the future based on current weather patterns. *Classification* attempts to model the observations to predict a discrete target variable, e.g. classifying a person based on his face image. In information retrieval, the observations are used to *rank* the data in order of certain preferences. Certain methods attempt to capture the general pattern underneath the data, e.g. Parzen window estimate to learn the underlying data distribution. A common theme in all these methods is to look for *special structures* in the unstructured raw data.

Learning methods can be broadly categorized as parametric and non-parametric

approaches. Parametric approaches assume a structure to the function to be estimated and uses observations to estimate the parameters of the assumed structure and thus the function. When there is prior information available about the model, the parametric model is a favorable choice.

Non-parametric methods do not assume any such structure for the function and generally "allow the data to speak for itself". They are very robust in modeling the non-linearities, and can be used when the parametric approaches fail (due to absence of prior knowledge of the model or due to improved robustness requirements).

Both these methods have their own advantages and disadvantages. The use of either of these methods for a particular application is determined by its effectiveness with the underlying data. Effectiveness of a learning method to a particular data depends on the nature of the attributes recorded and the target application.

1.2 Large scale data

The ease of data collection has led to a surge of the number of observable attributes and the low cost of data storage has resulted in a large number of samples being stored. This results in the availability of *tall fat* data for learning. Internet bigwigs like Google handle several petabytes of data daily and the data at this scale offer sufficient information to aid in better learning models. However, with the data at such scale, the scalability of any chosen learning approach becomes as important as its effectiveness to the underlying data. Non-parametric methods are computationally much more expensive than parametric methods and therefore

require special focus.

The scalability of an approach can be addressed either via algorithmic improvements or via parallelization. Algorithmic improvements approximate the underlying problem or cast them in a different framework, and thereby reduce the overall asymptotic complexity. Parallelization techniques make use of modern multi-core architecture (e.g. OpenMP, GPU/CUDA) or distributed systems (e.g. Hadoop, MPI) to enable scalability to large datasets.

1.3 Curse of dimensionality

In large scale data, several noisy dimensions are also encountered in most cases necessitating the need for denoising the data before applying any learning method. Further, for most machine learning tasks even though the data is very high dimensional, the true intrinsic dimensionality is typically very small. That is, the number of actual dimensions required for the target modeling/prediction is much lesser than those observed/recorded. All these have led to the study of several dimensionality reduction techniques and subspace modeling.

Dimensionality reduction techniques can be supervised or unsupervised. The popular principal component analysis (PCA) is an unsupervised technique that learns projects where the data variability is maximum. PCA is very used to remove noisy directions from the data. However, it leads to the same projections irrespective of the target application. Often, an application specific subspace is desirable for better learning, Fisher discriminant analysis, canonical correlation analysis, par-

tial least squares are some techniques that aid in such supervised dimensionality reduction.

1.4 Choice of learning method

The choice of a particular learning method is dictated by the characteristics of the underlying data and the target application. If the data is well-correlated and low-dimensional, any prior knowledge available on the data can be used to build a parametric model. In the absence of prior knowledge, non-parametric methods can be used. If the data is high-dimensional, PCA based dimensionality reduction is often the first step used. Alternately, if the precise target to be modeled is known, supervised dimensionality reduction can be used directly to achieve more correlated projections. The choice of learning methods can be made as before after the appropriate dimensionality reduction. Some dimensionality reduction techniques such as Canonical Correlation Analysis (CCA), Partial Least Squares (PLS) can be directly extended to a regression / classification technique, which is handy in some applications as well.

1.5 Thesis focus

The primary focus of this thesis is to develop effective and scalable learning methods for geostatistics and speaker recognition. The scalability of the methods that we develop/explore is addressed via graphical processors (GPUs). We initially focus on the computational bottlenecks in kernel based learning methods. We di-

vide the computational bottlenecks into three main categories and accelerate each of them on a GPU [95, 101]. Given the GPU acceleration, we improve the computational performance in a kernel regression via fast preconditioners in a flexible Krylov solver [97]. The combined framework provides a scalable approach to solve kernel regression.

1.5.1 Geostatistics

In geostatistics, data is often collected in scattered locations. However, for practical analysis, the data is required in a regular grid. In some cases, the observations might be missing due to instrument malfunctions or other reasons. Kriging[42] is a very popular geospatial interpolation technique for effective data reconstruction. We draw connections between kriging and kernel regression and extend the GPU accelerations previously developed with an optimized usage of GPU texture memory to improve scalability for the data in kriging [98]. We also design a gradient descent based parameter estimation technique for kriging. The combined framework is used successfully to reconstruct ocean color satellite data over the Chesapeake bay and Pacific ocean.

1.5.2 Speaker recognition

Speaker recognition is the task of verifying the identity of a speaker based on his speech samples. We focus on a text-independent speaker recognition framework. Features in speaker recognition are extracted from overlapping time frames.

Because of a text-independent framework, these features can be seen as samples from a target distribution and the problem of recognition becomes evaluating the probabilistic distance between distributions. We develop a Rényi entropy based distance using Parzen window estimates [96] and utilize the GPU accelerations to develop a fast probabilistic distance[94] and illustrate its application to speaker recognition[99]. However, its performance is limited due to lack of generalizability with limited training data. To address this, we explore a partial least square (PLS) framework for speaker recognition for better generalization [104] and extend it to its kernelized framework to improve recognition robustness using i-vectors [100]. The PLS framework was accelerated on a GPU [102] and its kernelized version utilizes GPUML for its acceleration.

1.6 Organization

This thesis is organized as follows. In Chapter 2, we introduce a generic GPU framework to solve several computational issues in kernel methods. In Chapter 3, we extend these ideas to a geostatistical interpolation technique, kriging. We also bring out the connection between kriging and Gaussian process regression, and utilize the connection to estimate prediction variance. In Chapter 4, we introduce a GPU-based fast preconditioner to efficiently solve the kernel regression with a flexible Krylov solver. In Chapter 5, we introduce the probabilistic distance based on the quadratic Rényi entropy and its kernelization. We illustrate its performance with a subset selection approach in a suite of learning and vision problems. We adapt the

kernelized Rényi distance (KRD) for speaker recognition in Chapter 6. While the performance of KRD for speaker recognition is promising, it fails against a state-of-the-art system. We therefore introduce a partial least squares (PLS) framework for speaker recognition that compares well with the state-of the-art systems in Chapter 7 and kernelize the framework for a more robust performance in Chapter 8. While the approaches developed in this thesis are targeted towards specific applications, the frameworks can be extended to many other problems as well. This is illustrated in Chapter 9 by extending the PLS framework to predict risk of loan defaults for a loan monitoring agency. Chapter 10 concludes the thesis with pointers to a few future directions.

Chapter 2

Graphical Processors for Machine Learning

During the past decades, several advances have been made in the field of machine learning and pattern recognition. Kernel machines are a particular class of learning approaches, that are very popular for their robustness. Several variants of kernel machines such as support vector machines, kernel density estimation, Gaussian process regression have been proposed and used successfully in many practical applications. However, the computational complexity of these are either quadratic or cubic, thus hindering their application to very large datasets. This chapter focuses on addressing these computational issues via the use of graphical processors. The core algorithms that will be discussed in this chapter are available as an open source software, GPUML [95, 101].

2.1 Computational bottlenecks in kernel machines

A core computation in many kernel approaches is the weighted summation of kernel functions

$$f(x_j) = \sum_{i=1}^{N} q_i K(x_i, x_j), \quad \mathbf{f} = \mathbf{K}\mathbf{q}, \tag{2.1}$$

which may also be treated as the product of a kernel matrix $\mathbf{K}$ with a vector $\mathbf{q}$. Here x_i is the d-dimensional observation. Typically, $f(x)$ needs to be evaluated at M points, resulting in an overall complexity of $\mathrm{O}(MN)$. By evaluating the

kernel function *on the fly*, the space complexity can be kept to O($M + N$). Other computations with the matrix $\mathbf{K}$, or its relatives, may also be sought, including solution of linear systems, eigen decomposition and others, and usually the complete matrix has to be stored in some of these cases, increasing the memory complexity to O(MN).

Existing approaches to accelerate kernel methods, either approximate the kernel summation/decomposition or parallelize them. Approaches like the Improved Fast Gauss Transform [119, 72] and dual-trees [51] evaluate kernel sums in linear time using efficient approximations. Message Passing Interface (MPI) on distributed clusters [54] and thread-based parallel approaches on graphical processing units (GPU)[105, 64] have been used to parallelize and speed up kernel machines. Most of the GPU based parallelization have primarily cast the underlying problems in terms of pixel and fragment shaders. With the emergence of CUDA (Compute Unified Device Architecture), it is possible to remove this additional overhead to better exploit the computational capabilities of the GPU. This has been used to accelerate the popular kernel machine, SVM in [107, 17]. Although CUDA based GPU algorithms have been used in some applications, a comprehensive work on the use of GPU for kernel machines has never been done and in this chapter, we try to address this by accelerating the following categories of kernel based algorithms using CUDA on GPU,

1. simple matrix-vector product involving kernel matrices (eg. for kernel density estimation)

2. solution of linear system of kernel matrices (eg. for kernel regression)

3. decomposition (like Cholesky, QR) of kernel matrices (eg. for spectral clustering)

We feel that the computational bottleneck in most of the kernel machines falls into one of these three categories. We propose approaches to accelerate each of these on a GPU and illustrate the speedup on applications like kernel density estimation, mean shift clustering, Gaussian process regression, ranking and kernel discriminant analysis.

This chapter is organized as follows. In section 2.2, we introduce the graphical processors and discuss their capabilities. In section 2.3, we discuss the mapping of various kernel problems on to the GPU and introduce the GPU-accelerated kernel matrix-vector product and matrix decomposition. In section 2.4, we present the various experiments performed to illustrate the speedups obtained in each case. Finally we provide our conclusions in section 2.5.

2.2 Graphical processors

Computer chip-makers are no longer able to easily improve the speed of processors, with the result that computer architectures of the future will have more cores, rather than more capable faster cores. This era of multicore computing requires that algorithms be adapted to the data parallel architecture. A particularly capable set of data parallel processors are the graphical processors, which have evolved into highly capable compute coprocessors. A graphical processing unit (GPU) is a highly

parallel, multi-threaded, multi-core processor with tremendous computational horsepower. In 2008, while the fastest Intel CPU could achieve only $\sim$ 50Gflops speed theoretically, GPUs could achieve $\sim$ 950Gflops on actual benchmarks [63]. Fig. 2.1 shows the relative growth in the speeds of NVIDIA GPUs and Intel CPUs (similar numbers are reported for AMD/ATI CPUs and GPUs); the FERMI architecture significantly improves these benchmarks further. Moreover, GPUs power utilization per flop is an order of magnitude better. GPUs are particularly well-suited for data parallel computation and are designed as a single-program-multiple-data (SPMD) architecture with very high arithmetic intensity (ratio of arithmetic operation to memory operations). However, the GPU does not have the functionalities of a CPU like task-scheduling. Therefore, it can efficiently be used to assist the CPU in its operation rather than replace it.

In November 2006, NVIDIA introduced *Compute Unified Device Architecture (CUDA)*[63], a parallel programming model that leverages the parallel compute engine in NVIDIA GPUs to solve general purpose computational problems. With CUDA, GPUs can be seen as a collection of parallel co-processors that can assist the main processor in its computations. The OpenCL initiative seeks to provide a similar non-proprietary API for general purpose GPU computing.

Fig. 2.2 shows how current GPU coprocessors appear to a user through CUDA. Each GPU has a set of multiprocessors, each with 8 processors. All multiprocessors communicate with a global memory, which can be as large as 4GB, and a constant/texture memory. More capable GPUs share more multiprocessors and more global memory. The 8 processors in each multiprocessor share a local shared mem-

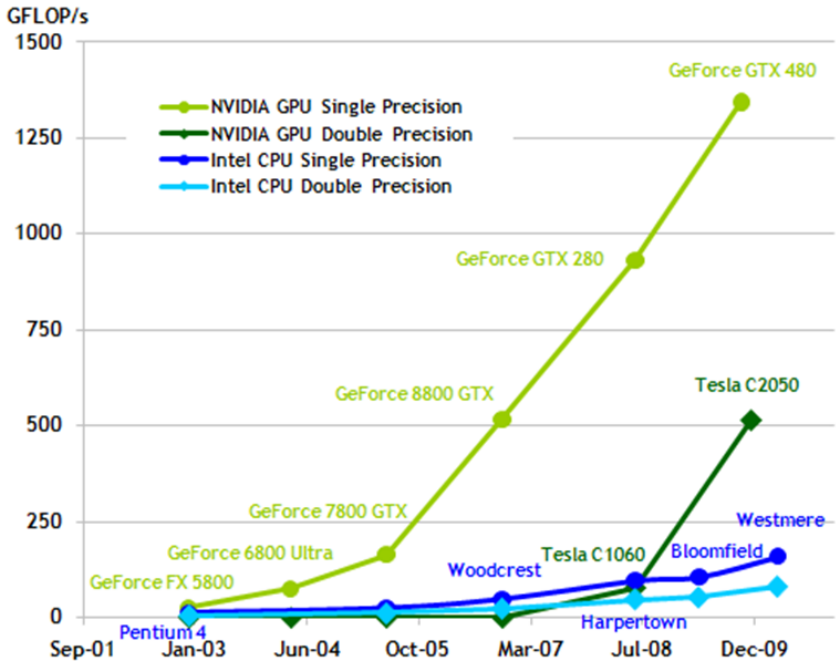


Figure 2.1: [color] *Growth in the CPU and GPU speeds over the last* 6 *years on benchmarks (Image from [63])*

ory and a local set of registers. The instructions in the GPU are designed to be executed as parallel threads on multiple data. Therefore, the computations are organized into grids, which are groups of thread blocks. A thread block is defined as a patch of threads that are executed on a single multiprocessor. A maximum of 512 threads can be housed in a single thread block. Each thread performs its operations independently and halts when a synchronization barrier is reached. *While GPUs can do double precision, most advantage is gained on single precision computations, and double precision is advised only when it is absolutely essential for algorithmic correctness.* Newer GPUs that are to be released in coming years relax this restriction.

The main processor (the *host*) controls the computations and provides the

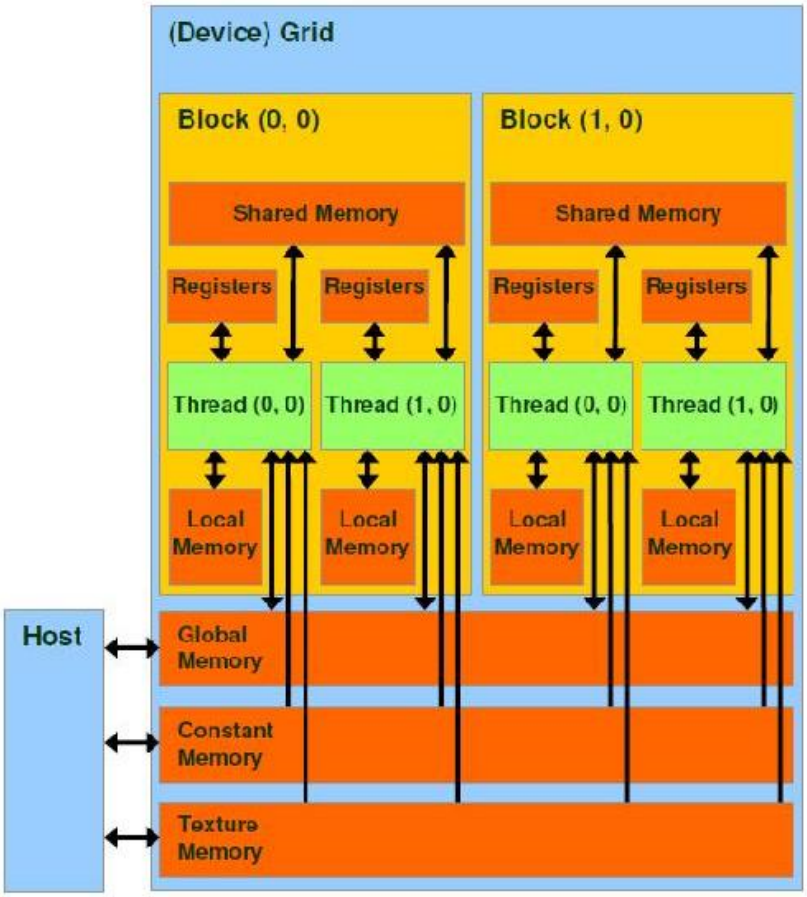


Figure 2.2: [color] *Logical organization of the GPU memories as seen through CUDA (Image from [63])*

data on which the GPU (the *device*) can work on. This data is generally transferred from the host memory to the device's global memory. The global memory is large enough to hold many of the large datasets usually encountered ($\sim$ 4GB on current GPUs). It is important to note that access times to different memories in the device are significantly different. Accesses to global memory are the most expensive and it takes approximately 400 clock cycles for one access. However, if each thread in a block accesses consecutive global memory locations, it takes less time than a random

access. This is referred to as *memory coalescing*. Accesses from the cached constant and texture memory, which can be written to from the host, are cheaper. Read and write local memory is provided by shared memory (which is shared between all processors in a multiprocessor), and per-processor register memory, and takes only as long as one instruction.

2.2.1 Addressing scalability with GPUs

The key difference between an efficient algorithm on a sequential processor and a graphics processor is that the former requires to have as little computation as possible while the latter needs to minimize memory access to and from the global memory. In other words, an efficient GPU algorithm should ensure a coalesced transfer of data from the global memory to the local shared memory, a parallelization strategy that results in most of the work being done on data that is in local registers or shared memory, and a well defined patterns of access to global memory.

2.3 Accelerating kernel methods on GPUs

2.3.1 Kernel summation:

We will refer to the data points x_i in Eq. (2.1) as *source* points, and the points at which the kernel sums are evaluated as *evaluation* points (in line with N-body algorithms, which have a similar computational structure). There are several ways each thread can be designed. One obvious parallelization approach is to assign each thread to process the effect of each source point; another one is to assign each thread

GPU based acceleration for kernel summation
Data: Source points $x_i, i = 1, \ldots, N$, evaluation points $y_j, j = 1, \ldots, M$
Each thread evaluates the sum corresponding to one evaluation point:
Step 1: Load evaluation point corresponding to the current thread in to a local register.
Step 2: Load the first chunk of source data to the shared memory.
Step 3: Evaluate part of kernel sum corresponding to source data in the shared memory.
Step 4: Store the result in a local register.
Step 5: If all the source points have not been processed yet, load the next chunk, go to Step 3.
Step 6: Write the sum in the local register to the global memory.

Table 2.1: *Data-parallel kernel summation on the GPU*

to process individual evaluation points independently.

If each thread is assigned to evaluate the effects of a particular source on all evaluation points, it would have to update the value at each evaluation point in the global memory, thus requiring a number of global memory writes, resulting in a memory inefficient algorithm. However, if each thread is assigned to evaluate a particular evaluation point, it would require only one global memory write per thread. This would however result in several global reads per thread. In this case the use of shared memory and registers can reduce the number of global accesses.

We assign each thread to evaluate the kernel sum on an evaluation point. If

there are N source points, then each thread would be required to read N source points from global memory. We reduce the total accesses to global memory by transferring source points to the shared memory. The shared memory is not large enough to house the entire set of data. So it is required to divide the data into chunks and load them according to the capacity of the shared memory (the number of source points that can be loaded to the shared memory is limited by its size and data dimension). The size of each chunk is set to be equal to the number of threads in the block, and each thread transfers one source element from the global memory to the shared memory, thus ensuring coalesced memory reads. The weights corresponding to each source is loaded to the shared memory in the same way. Once the source data is available in the shared memory, all threads update the kernel sums involving the source points in the shared memory.

In order to further reduce the global memory accesses, we use local registers for each evaluation point and the evaluation sum. Once all the source data are evaluated, the sum in the register is written back to global memory. The algorithm is summarized in Table 2.1. If d is the dimension of the data points, then we use $d+1$ shared memory location per thread (one source point and its kernel weight) and $d+1$ registers per thread (one evaluation point and the corresponding kernel sum). The proposed approach is generic and can be extended to any kernel, an important distinction from the CPU based approximation algorithms [51, 119, 72, 58].

2.3.2 Accelerating iterative algorithms:

Several kernel machines involve solution of linear or least square systems with kernel matrices, or computation of a few eigenvalues and eigenvectors of a kernel matrix. Iterative approaches are used for problems of this type. These include conjugate gradients [34] for Gaussian process regression, power iterations for eigenvalue computations, more sophisticated Krylov and Arnoldi methods, and others. In each case, the core computation per iteration is the matrix vector product, with the computation of a residual or error term. We discuss individual cases in the experimental section. As far as GPU implementations are concerned, accelerations are achieved by having the above sum evaluated in the iterative procedure. Better speedups can be obtained if the data between iterations are allowed to stay on the GPU, thus avoiding data transfers between host and device. The other way to accelerate these algorithms is to reduce the number of iterations via techniques such as preconditioning. This discussion is deferred till Chapter 4.

2.3.3 Accelerating kernel matrix decompositions:

Several matrix decompositions are already available on the GPU [113] and here, the strategy is to use CPU algorithms, with part of the computation performed on the GPU. To accelerate kernel methods, these libraries can be used as is, similar to the way Lapack and other libraries are used to accelerate CPU versions of kernel methods. Given the training and test data, the kernel matrix needs to be constructed before decomposition. Constructing the matrix has a computational

complexity of $O(dN^2)$ in most kernels, d being the dimension of the input data. For higher dimension ($d > 50$) the matrix construction cost can become as significant as the decomposition itself, because of the availability of optimized implementations of the decomposition algorithms. However, if the structure of the kernel matrices is utilized, the matrix construction can also be parallelized on the GPU. The matrix decomposition algorithms return a matrix which requires $O(N^2)$ memory, and memory requirements in these approaches cannot be reduced by generating the kernel matrices on-the-fly as in the kernel summation.

We construct the matrix on the GPU and utilize this matrix for decomposition using approaches in [113]. The proposed approach is summarized in Table 2.2. We load a chunk of the training and test points to the shared memory and generate the block of the kernel matrix that involving these training/test points in the current thread block. This is repeated across all the thread blocks to construct the entire kernel matrix, which can be used for matrix decomposition.

Testing these accelerated frameworks for very large datasets is close to impossible. We shall address this issue in Appendix A via use of random sampling with Chernoff bounds for small-scale testing of large-scale problems. In our subsequent experiments, we shall use moderately sized datasets for evaluating our accelerated frameworks.

Accelerated kernel matrix construction for matrix decomposition
Data: Source points $x_i, i = 1, \ldots, N$, evaluation points $y_j, j = 1, \ldots, M$
Each thread evaluates one element of the kernel matrix
Step 1: Load the source points from global memory into the shared memory.
Step 2: For large data dimension which can not fit into shared memory, divide the each source vector into several chunks of constant size and load them consecutively.
Step 3: Compute the distance contribution of the current chunk in a local register, and load the next chunk. Repeat this until the complete dimension is spanned.
Step 4: Use the computed distance for evaluating the matrix entry.
Step 5: Write the final computed kernel matrix entries into global memory.

Table 2.2: *Data parallel kernel construction on the GPU*

2.4 Experiments

We tested the GPU accelerated kernel methods with three classes of problems. The first problem class used the accelerated summation on different kernels and tested the speedup on a synthetic data. Further we extended our approach to speed-up kernel density estimation. We also compare GPU based Gaussian kernel summation with a linear algorithm, FIGTREE [58]. In the second problem class, we looked at different iterative approaches which employ the kernel summation over each iteration. Finally we look at algorithms that use kernel matrix decompositions

and accelerate them using our acceleration (Table 2.2).

2.4.1 Host and device:

In all experiments the host processor is an Intel Xeon Quad-Core 2.4GHz with 4GB RAM. The GPU is a Tesla C1060 which has 240 cores arranged as 30 multiprocessors. It has 4GB of global memory, 16384 registers per thread block and 16kB shared memory per multiprocessor.

For all experiments, GPU codes were written in CUDA and compiled with Matlab linkages. Similarly, the CPU codes were written in C++ with Matlab linkages. This allowed for convenient execution of machine learning algorithms.

2.4.2 Experiment1 - Kernel summations:

We accelerated widely-used kernels namely the Gaussian (Eq. 2.2), Matern (Eq. 2.3), periodic (Eq. 2.4) and Epanechnikov (Eq. 2.5) kernels given by,

$$K(x_i, x) = s \times \exp\left(-\frac{(d(x_i, x))^2}{2}\right), \tag{2.2}$$

$$K(x_i, x) = s \times (1 + \sqrt{3}d(x_i, x)) \times \exp(-\sqrt{3}d(x_i, x)), \tag{2.3}$$

$$K(x_i, x) = s \times \exp\left(-2\sin^2(\pi * d(x_i, x))\right), \tag{2.4}$$

$$K(x_i, x) = s \times (1 - d(x_i, x)^2) \times 1(d(x_i, x) < 1), \tag{2.5}$$

where $d(x_i, x)$ is the Euclidean distance between the points x_i and x, s is a scaling parameter and $1(d(x_i, x) < 1)$ is an indicator function. Also there is a bandwidth h

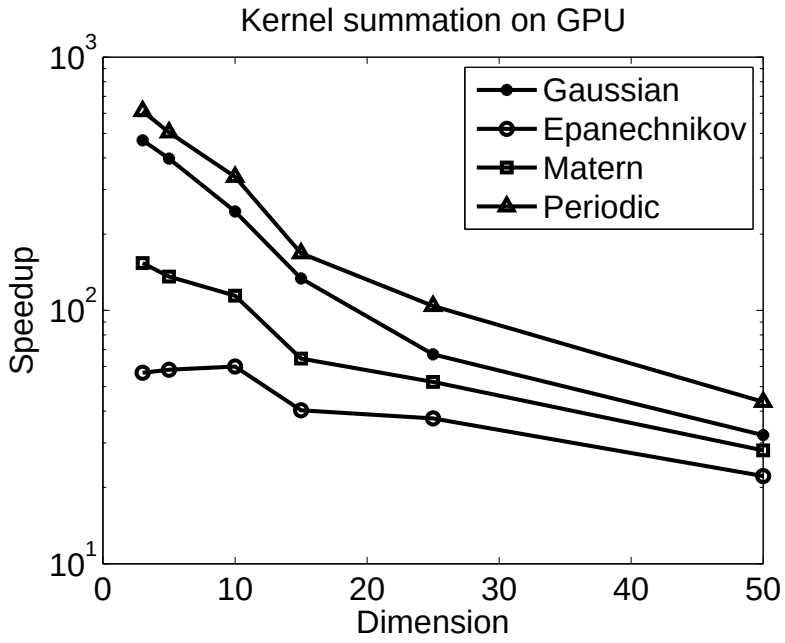


Figure 2.3: *Speedup obtained on a GPU for various kernel summations for a data size of* $10,000$. *The mean absolute error between the CPU and GPU based summation in all the cases were less* 10^{-5}.

associated with distance $d(.,.)$ such that,

$$d(x_1, x_2) = \sum_{k=1}^{d} \frac{\|x_{1,k} - x_{2,k}\|^2}{h^2}. \tag{2.6}$$

The synthetic data for this experiment were generated by choosing a random number between 0 and 1 uniformly for each dimension of the source and evaluation points. Datasets of varying dimensions are generated in this fashion. The resulting speedup for each kernel for a $10,000$-size data is shown in Fig. 2.3. The speedup obtained is lower for a simple kernel like Epanechnikov, but as the complexity of the kernel increases, the speedup obtained is significant, like for periodic/Gaussian. This can be attributed to the fact that for simpler kernels, the data transfer time is more dominant than for a complex kernel. As the dimension increases, the number

of threads that can be accommodated on each processor is reduced to fit the data in the shared memory, and hence there is a reduction in the speedup.

2.4.2.1 Comparison with FIGTREE:

There are several approximation algorithms that evaluate the kernel summations in linear time, for example, FIGTREE [58] for the Gaussian kernel. In spite of the speedups obtained by the GPU-based approach, the asymptotic dependence on data size is still $O(N^2)$. Therefore, a linear approach like FIGTREE [58] will outperform it at some point. In order to explore this, we compared the performance of our GPU algorithm with FIGTREE which combines two popular linear approximation algorithms, Improved Fast Gauss Transform (IFGT) [119, 72] and tree based approaches, and automatically chooses the fastest method for a given data. We expected FIGTREE to beat our implementation at some point, but this was observed only for a data size greater than $128,000$ as seen in Fig. 2.4.

Although the linear approach eventually outperforms the GPU version, the performance of these approaches are found to have great dependence on data dimensions and kernel bandwidth. Also, these approaches are restricted to the Gaussian kernel and require a fixed bandwidth over the data. In contrast, our GPU based approach can be used with any kernel for varying bandwidths. We compared the performance of our algorithm with FIGTREE for various dimensions at a different kernel bandwidth and the results are shown in Fig. 2.5. It can be seen that the speedup obtained by the GPU approach over FIGTREE increases with data

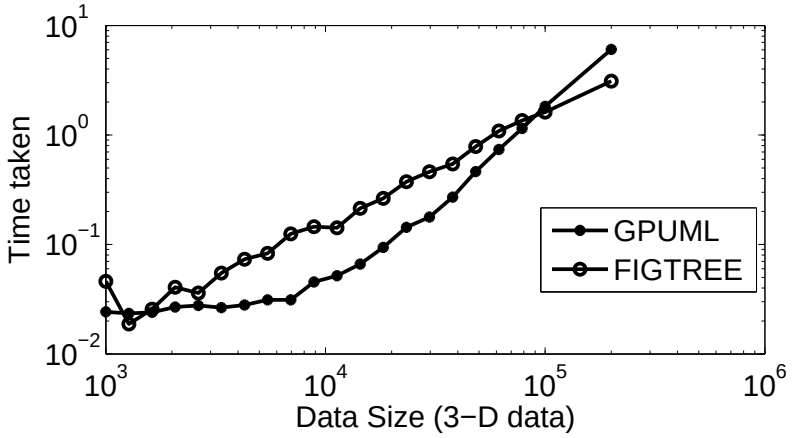


Figure 2.4: *Performance of GPUML for Gaussian kernel summation compared to FIGTREE[58], the linear algorithm for various data sizes at 3-dimensions*

dimensions.

2.4.2.2 Kernel Density Estimation (KDE):

KDE (or Parzen window based density estimation) is a non-parametric way of estimating the probability density function of a random variable. Given a set of observations $D = \{x_1, x_2, \cdots, x_N\}$, the density estimate at a new point x is given by

$$f(x) = \frac{1}{Nh} \sum_{i=1}^{N} K\left(\frac{x - x_i}{h}\right). \tag{2.7}$$

Two kernels widely used for density estimation are the Gaussian kernel (Eq. 2.2) and the Epanechnikov kernel (Eq. 2.5). We performed KDE based on GPU acceleration on the 15 normal mixture densities in [56] and compared performance with a direct approach. The error between the two approaches was less than 10^{-6} for each distribution. The results are tabulated in Table 2.3.

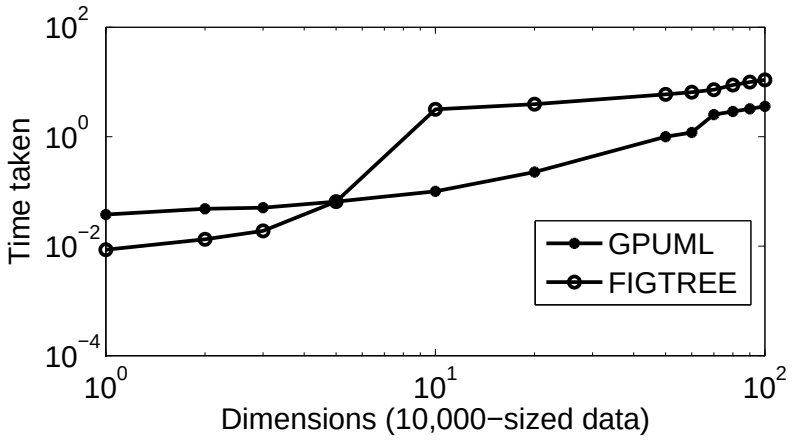


Figure 2.5: *Performance of GPUML for Gaussian kernel summation compared to FIGTREE[58], the linear algorithm for various data dimensions*

Mean CPU time for Gaussian kernel	25.144s
Mean GPU time for Gaussian kernel	0.022s
Mean absolute error between estimates	$\sim 10^{-7}$
Mean CPU time for Epanechnikov kernel	25.117s
Mean GPU time for Epanechnikov kernel	0.011s
Mean absolute error between estimates	$\sim 10^{-7}$

Table 2.3: *Performance of kernel density estimation on the* 15 *normal mixture densities in [56] for a data size of* 10000

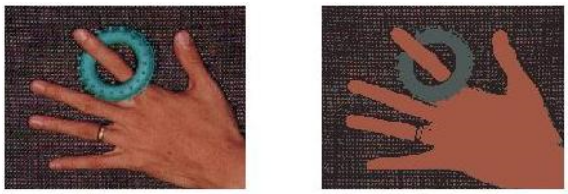


Figure 2.6: [color] *An example of mean shift clustering on a color image; Left: Original Image; Right: Segmented image*

2.4.3 Experiment2 - Iterative approaches using kernel summations:

In this experiment, we explored different algorithms, that uses the kernel summation (Eq. 2.1) within an iterative algorithm like conjugate gradient.

2.4.3.1 Mean Shift Clustering:

Mean shift clustering [19] is a non-parametric clustering approach based on kernel density estimation. It involves running a ***gradient ascent*** over the kernel density estimate in order to move each data point towards the local mode. Finally, it returns the set of modes (centers) to which each data point converges. A typical result of mean-shift based clustering on an image is shown in Fig. 2.6.

In this experiment, we apply the GPU-based density estimates over each iteration of the gradient ascent to speed up the mean shift clustering approach in [19]. For the image in Fig. 2.6, it was observed that the naive direct implementation took almost 13.5 hours, whereas the corresponding GPU implementation took only 35 seconds for clustering.

2.4.3.2 Optimal Bandwidth Estimation:

Determining the bandwidth h of the kernel in density estimation is of paramount importance for the performance of the estimator [87]. For a Gaussian kernel, there have been several approaches to evaluate the optimal bandwidth for given data points. We looked at the plug-in approach in [85] and accelerated it using the GPU. The key computation in the bandwidth selection approach in [85] is the evaluation of a weighted sum of the *GaussianDerivative* kernel,

$$K(x_i, x) = s \times H_r(d(x_i, x)) \times \exp\left(-\frac{(d(x_i, x))^2}{2}\right), \tag{2.8}$$

where H_r is the r^{th} Hermite polynomial. We used a GPU based *GaussianDerivative* kernel summation with a ***Levenberg-Marquardt based non-linear least squares*** to solve for the bandwidth of the Gaussian kernel as in [85] and the results are shown in Table 2.4.

Mean CPU time	390.847s
Mean GPU time	0.369s
Mean abs error	$< 10^{-5}$

Table 2.4: *Performance on the optimal bandwidth estimation problem on the* 15 *normal mixture densities in [56] for a data size of* 10000

2.4.3.3 Gaussian Process Regression:

Gaussian process regression is a probabilistic kernel regression approach which uses the prior that the regression function ($f(X)$) is sampled from a Gaussian pro-

cess. For regression, it is assumed that a set of datapoints $D = \{X, y\}_{i=1}^{N}$, where X is the input and y is the corresponding output. The function model is assumed to be $y = f(x) + \epsilon$ where ϵ is a Gaussian noise with variance σ^2. Rasmussen et al. [69] use the Gaussian process prior with a zero mean function and has a covariance function defined by a kernel $K(x, x')$ which is the covariance between x and x', i.e. $f(x) \sim GP(0, K(x, x'))$. They show that with this Gaussian process prior, the posterior of the output y is also Gaussian with mean m and covariance V given by,

$$\begin{aligned} m &= k(x*)^T (K + \sigma^2 I)^{-1} y \\ V &= K(x_*, x_*) - k(x_*)^T (K + \sigma^2 I)^{-1} k(x_*) \\ k(x_*) &= [K(x_1, x*), K(x_2, x*) \ldots, K(x_N, x*)] \end{aligned}$$

where $x*$ is the input at which prediction is required and . Here m is the prediction at $x*$ and V the variance estimate of prediction. Some popular kernels used with Gaussian process regression are Gaussian (Eq. 2.2), Matern (Eq. 2.3)and periodic kernels (Eq. 2.4).

The parameters of the kernels s and h are called the *hyperparameters* of the Gaussian process and there are different approaches to estimate these [69]. Given the hyperparameters, the core complexity in Gaussian processes involves solving a linear system involving the kernel covariance matrix and hence is $O(N^3)$. Gibbs et al. [34] suggest a *conjugate gradient* based approach to solve the Gaussian process problem in $O(kN^2)$, k being the number of conjugate gradient iterations. In each iteration of the conjugate gradient solver, the key computation is a weighted summation of the covariance kernel functions. In this experiment, we used our GPU based kernel

Dataset	dxN	CPU Time (s)	GPU Time (s)
Diabetes	2x43	0.0473	0.1639
Abalone	7x4177	235.8	0.79
Bank7FM	8x4499	631.2	2.19
CompAct	22x8192	1884.2	9.3
PumaDyn8NH	8x4499	467.69	1.72
Stock	9x950	13.34	0.27

Table 2.5: *Performance on Gaussian process regression with standard datasets; d denotes the input dimension and N the size of the input data. The mean absolute error in each case was less than* 10^{-5} *for a the Gaussian covariance (kernel) (Eq. 2.2)*

summation to speed up each iteration of the conjugate gradient. Table 2.5 shows the performance of Gaussian process regression on various standard datasets [110] for a Gaussian kernel.

2.4.3.4 Learning a Ranking function:

In information retrieval, a ranking function is a function that ranks data matching a given search query point according to their relevance. In order to rank the data, a preference relation needs to be learned, which is given by the ranking function. A ranking function f maps a pair of data-points to a score value, which

Dataset	dxN	Raykar et al.[71]	GPU
Auto	8x392	0.7499s	0.5280s
CA Housing	9x20640	105.2s	27.82s
CompAct	22x8192	5.67	5.56s
Abalone	8x4177	9.838s	5.003s

Table 2.6: *Performance of WMW-statistic based ranking, GPU based approach vs the linear algorithm in [71]*

can be sorted for ranking. There are several approaches to learn the ranking function for a given dataset. We particularly looked at the preference graphs based approach in [71].

Raykar et al. [71] maximize the generalized Wilcoxon-Mann-Whitney (WMW) statistic [66] using a *non-linear conjugate gradient* approach to learn the ranking function. Instead of maximizing the WMW statistic, they use a continuous surrogate based on the log-likelihood, thus maximizing a relaxed statistic. They use a sigmoid function to model the pair-wise probability, and approximate it using an *erfc* function. They thus reduce the core computation of the ranking problem to the evaluation of a weighted sum of erfc functions. A linear algorithm for the summation of the erfc functions is proposed in [71], which is then used for the efficient learning of the ranking function.

We use GPU based summation of *erfc* functions for learning the ranking function and compare it with the linear approach in [71] on the datasets used in [71] and the results are shown in Table 2.6. It can be seen that our approach consistently

outperforms the approach in [71]. Note that the approach in [71] is linear in computational complexity, whereas our approach is quadratic and still outperforms the linear approach for large datasets.

2.4.4 Experiment3: Matrix decompositions

Often, in many algorithms, it is required to perform LU, QR or Cholesky decomposition of the kernel matrices, for example [11] and a direct decomposition will have a cubic complexity. The GPU-based summation cannot be used in these scenarios, but there are many algorithms for performing efficient matrix decompositions accelerated on GPUs. In this experiment, we discuss these approaches for kernel matrices.

Volkov et al. [113] claim that their implementation is the fastest LU, QR and Cholesky decomposition as of 2008. We adapted their approach for kernel matrices in this experiment. As the dimension of the input data increases, the construction of the kernel matrices becomes the dominant part of the decomposition, due to the high efficiency of these algorithms. In order to illustrate this, we performed Cholesky and QR decompositions of kernel matrices for data of various dimensions for a $10,000$-size synthetic data (generated as before). We decomposed a Gaussian kernel matrix on the GPU using [113], compared the performance for the matrix constructed on the CPU with the GPU counterpart, and the results are shown in Fig. 2.7. As the data dimension increases, the speedup obtained also increases. This suggests that for large dimensions, matrix construction takes significant time

and hence the proposed approach would increase speedup.

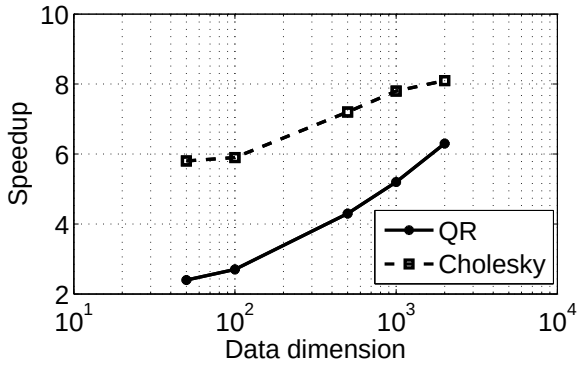


Figure 2.7: *Speedup obtained for Cholesky and QR decomposition of a Gaussian kernel matrix, based on [113] on a* $10,000$*-size data (size of the kernel matrix) for various dimensions of the input data; the speedups reported here are for the matrix construction on GPU using Table 2.2 and decomposition using [113] against the matrix construction on CPU and GPU decomposition using [113]. The mean absolute error in each case was less than* 10^{-4}.

2.4.4.1 Spectral Regression for Kernel Discriminant Analysis:

Linear discriminant analysis (LDA) is a statistical projection approach, where the projections are obtained by maximizing the inter-class covariance, while simultaneously minimizing the intra-class covariance. For non-linear problems, the LDA is performed on the kernel space and is termed as Kernel discriminant analysis (KDA). KDA requires the eigen decomposition of the kernel matrix and hence is computationally expensive for large datasizes. Cai et. al [11] address this problem by casting the eigen decomposition problem as a spectral regression and have proposed a spectral-regression based KDA (SRKDA). At the core of SRKDA is

a Cholesky decomposition of the kernel matrices, this has resulted in a 27 times theoretical speedup of SRKDA over KDA.

However, the Cholesky decomposition remains the core computational task of SRKDA, and its cubic complexity is still computationally expensive for large datasizes. We address this problem using the proposed kernel matrix decomposition. We performed SRKDA-based decomposition on the SIFT features extracted from the 10 classes of the CalTech-101 dataset [30]. The results are tabulated in Table 2.7. It is evident that there is a significant improvement in the performance compared to a direct implementation.

DataSize	Direct	GPU
1000	0.61s	0.2830s
2500	4.41s	2.1337s
5000	22.06s	12.467s
7500	60.30s	37.28s

Table 2.7: *SRKDA [11] - Comparison between the GPU implementation and a CPU implementation on Caltech-101 dataset [30]; mean absolute error (measured on the projection of a test data of size* 100*) in each case was* $\sim 10^{-5}$

2.5 Conclusions

In this chapter, we have looked at accelerating popular kernel approaches on the GPU. We have reported the speedups obtained for the summation and decom-

position of various kernels on GPUs. Our approaches are not just limited to the kernels reported and can be extended to any generic kernel. Further, we have shown the improvement in performance with different kernel machines like kernel density estimation, Gaussian process regression, learning a Ranking function and kernel discriminant analysis using spectral regression. We have also compared our performance with a linear algorithm [58]. With the increasing speeds in GPUs compared to the CPUs (as shown in Fig. 2.1), the performance can only improve further and can provide effective solution to computation bottlenecks in various kernel machines. We have made these algorithms in CUDA with Matlab linkages available under LGPL as an opensource[1] and we hope to keep evolving the library.

In subsequent chapters, we shall extend these GPU frameworks to more targeted applications like kriging and speaker recognition.

[1] www.umiacs.umd.edu/users/balajiv/GPUML.htm

Chapter 3

Spatio-temporal kriging for geospatial data reconstruction

In the last chapter, we discussed the utilization of GPUs to accelerate different kernel based algorithms. In this chapter, we introduce the geospatial interpolation technique, kriging and bring out its connection with the Gaussian process regression (introduced in the last chapter). We then specialize the GPU acceleration used with Gaussian process regression for kriging by better utilization of the GPU's texture memory. The resulting kriging framework is also used with a modified gradient descent algorithm to estimate the kriging parameters via cross-validation.

3.1 Geospatial data reconstruction

Sensors deployed on satellites are often used to collect enviromental data where a direct measurement is expensive or difficult. For example, satellite images allow general studies of the sea surface characteristics as well as studies at depth that present a signal at the surface, such as from internal waves. These images are widely used due to their vast spatial coverage and precise observation. Several kinds of data are measured by satellites using sensors that span the electromagnetic spectrum. Sensors working in the visible and infrared frequencies (e.g. to measure ocean color) are affected by cloud cover. Instrument malfunctions or the variable satellite orbital paths lead to missing values in the observed data. A complete

data set is important for initializing many meteorological models or data analysis. This problem is receiving significant attention because a better understanding of climate variability due to human and geological factors is required, and because of the importance such data have in planning during an era of considerable climate change.

Techniques like spline data interpolation [28] and inverse methods [39, 44] have been explored for the reconstruction of the missing satellite observations. An alternate self-consistent model has been proposed in [4] and developed as the Data INterpolating Empirical Orthogonal Functions (DINEOF); and has been used for several data reconstructions [1, 2]. In this chapter, we reconsider the use of geostatistical kriging [42] as an alternate data reconstruction approach, and exploit its connection to the modern statistical technique of Gaussian process regression (GPR) [70] to provide an approach to estimate the variance of the reconstruction.

Kriging [42] is a group of geostatistical techniques to interpolate the value of a random field (e.g., the elevation, z, of the landscape as a function of the geographic location) at an unobserved location from observations of its value at nearby locations. It belongs to a family of linear least squares estimation algorithms that are used in several geostatistical applications. It has its origin in mining application, where it was used to estimate the changes in ore grade within the mine [49]. Kriging has since been applied in several scientific disciplines including atmospheric science, environmental monitoring and soil management. However, kriging has a cubic computational cost, that can be detrimental for large datasets. We mitigate this problem using an iterative formulation that can be implemented efficiently on

multi-core graphical processors.

This chapter is organized as follows. In Section 3.2, the kriging framework is introduced and its relation to Gaussian process regression is discussed. In Section 3.3, the computational cost of kriging is addressed via the use of graphical processors and a modified gradient descent is introduced to estimate the parameters of kriging in Section 3.4. We reconstruct the SeaWiFS chlorophyll concentration recorded over Chesapeake bay and Pacific ocean using the developed framework and is compared to the reconstruction with DINEOF in Section 3.5.

3.2 Kriging

Kriging is a popular interpolation technique used with geostatistical data. It has linear and non-linear variants. Linear kriging estimates the unknown data as a linear combination of known data. Simple kriging, ordinary kriging and universal kriging are linear variants. The nonlinear indicator kriging, log-normal kriging and disjunctive kriging were developed to account for non-linearities in the models. Moyeed et al. [59] show that the performance of linear and non-linear kriging are comparable except in the use of skewed data where non-linear kriging performs better. We shall restrict the discussion here to simple kriging, although our approaches can be extended to the other variants as well. Simple kriging is statistically the *best linear estimator*. It is termed the *best* because it attempts to minimize the residual variance.

3.2.1 Formulation

Let the data be sampled at N locations $(x_1, x_2, \ldots, x_N)$, and the corresponding values be $(v_1, v_2, \ldots, v_N)$. The value $\hat{v_j}$ at an unknown location $\hat{x_j}$ is estimated as a weighted linear combination of v's, given by $\tilde{v}_j = \sum_{i=1}^{N} w_i^j v_i$. Here, $\tilde{v}_j$ is the estimate, and let $\hat{v}_j$ be the actual (unknown) value at $\hat{x}_j$. The residue $r_j = \tilde{v}_j - \hat{v}_j$ and the residual variance is then given by,

$$var(r_j) = cov\left(\tilde{v}_j, \tilde{v}_j\right) - 2cov\left(\tilde{v}_j, \hat{v}_j\right) + cov\left(\hat{v}_j, \hat{v}_j\right) \tag{3.1}$$

The first term can further be simplified as,

$$\begin{aligned} cov\left(\tilde{v}_j, \tilde{v}_j\right) &= cov\left(\sum_{i=1}^{N} w_i^j v_i, \sum_{i=1}^{N} w_i^j v_i\right) \\ &= \sum_{i=1}^{N}\sum_{k=1}^{N} w_i^j w_k^j cov\left(v_i, v_k\right) = \sum_{i=1}^{N}\sum_{k=1}^{N} w_i^j w_k^j \mathbf{C}_{ik}, \end{aligned}$$

where $\mathbf{C}_{ik} = cov(v_i, v_k)$. Similarly, the second term becomes,

$$cov\left(\tilde{v}_j, \hat{v}_j\right) = cov\left(\left(\sum_{i=i}^{N} w_i^j v_i\right), \hat{v_j}\right) = \sum_{i=1}^{N} w_i^j \mathbf{C}_{ij}^*,$$

where $\mathbf{C}_{ij}^*$ indicate the covariance between known location i and unknown location j. Finally, assuming that the random variables have the same variance σ_v^2, the third term can be expressed as $cov\left(\hat{v_j}, \hat{v_j}\right) = \sigma_v^2$.

For simple kriging, it is required to find w^j by minimizing $var(r_j)$ with respect to w^j. This can be written as the minimization of the penalized cost function,

$$J = \sum_{i=1}^{N}\sum_{k=1}^{N} w_i^j w_k^j \mathbf{C}_{ik} + \sum_{i=1}^{N} w_i^j \mathbf{C}_{ij}^* + \sigma_v^2.$$

Taking the partial derivative w.r.t w_i^j,

$$\frac{\partial J}{\partial w_i^j} = 2\sum_{k=1}^{N} w_k^j \mathbf{C}_{ik} - 2\mathbf{C}_{i\hat{j}}$$

and setting it to 0,

$$\begin{pmatrix} \mathbf{C}_{11} & \dots & \mathbf{C}_{1N} \\ \vdots & \ddots & \vdots \\ \mathbf{C}_{N1} & \dots & \mathbf{C}_{NN} \end{pmatrix} \begin{pmatrix} w_1^j \\ \vdots \\ w_N^j \end{pmatrix} = \begin{pmatrix} \mathbf{C}_{1j}^* \\ \vdots \\ \mathbf{C}_{Nj}^* \end{pmatrix}$$

$$\Rightarrow \mathbf{C}\mathbf{w}^j = \mathbf{C}_j^* \tag{3.2}$$

In order to obtain the kriged output at M locations, Eq. (3.2) needs to be solved at each of these locations, resulting in the system,

$$\mathbf{v}^* = \mathbf{v}^T \mathbf{C}^{-1} \mathbf{C}^*, \tag{3.3}$$

where,

$$\mathbf{v}^* = \begin{pmatrix} \tilde{v}_1 \\ \vdots \\ \tilde{v}_M \end{pmatrix}, \mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_N \end{pmatrix}, \mathbf{C}^* = \begin{pmatrix} \mathbf{C}_1^* \\ \vdots \\ \mathbf{C}_M^* \end{pmatrix}^T$$

3.2.2 Covariance functions

The covariances $\mathbf{C}_{ij}$ can either be specified by a standard function or by statistically evaluating them at each location. The latter approach incurs huge storage costs for large datasets and is not easy to compute when reconstruction is required at an unseen location. A functional form of covariance is preferred in these cases. The covariance function is generally chosen to reflect prior information. In the absence of such knowledge, the Gaussian function is the most widely used covariance[42],

$$\mathbf{C}_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{h^2}\right). \tag{3.4}$$

Another advantage with a functional representation is that it is possible to krige by computing the matrix C on-the-fly thus saving on memory. We shall use this here.

3.2.3 Gaussian process regression & kriging

We can transform the above kriging problem into Gaussian process regression, and improve the efficiency. Evaluation of a single set of weights w^j for a given location j requires the solution of the system in Eq. (3.2) and has a computational complexity of $\mathrm{O}(N^3)$, N being the number of samples. Further, to get the weights at M locations, the complexity increases to $\mathrm{O}(MN^3)$. For $M \approx N$, the asymptotic complexity is $\mathrm{O}(N^4)$, this is undesirable for large N. Without loss of generality, the system in Eq. (3.3) can be transposed,

$$\mathbf{v}^* = \mathbf{C}^*\mathbf{C}^{-1}\mathbf{v}. \tag{3.5}$$

Now, the kriging comprises of solving a linear system followed by a covariance matrix-vector product; thus resulting in a complexity of $\mathrm{O}(N^3 + N^2)$. This is an order reduction from the original formulation. Note that, the weights are not evaluated explicitly, and so the storage is also avoided. The two steps of the new formulation are [70]

$$\text{Solve for } \mathbf{y},\ \mathbf{C}\mathbf{y} = \mathbf{v}, \quad \mathbf{v}^* = \hat{\mathbf{C}}^{*\mathbf{T}}\mathbf{y} \tag{3.6}$$

Such a formulation makes kriging similar to the training and prediction in Gaussian process regression [69].

3.2.4 Evaluation of the variance of the estimate

The Gaussian process similarity allows for the definition of a variance (Σ_j) [69] of the estimate at the j^{th} kriged location given by,

$$\Sigma_j = \mathbf{C}^*_{jj} - \mathbf{C}^{*T}_j \mathbf{C} \mathbf{C}^*_j \tag{3.7}$$

Note that the cost of each variance estimate is $O(N^3)$ per estimate.

3.3 Accelerated kriging

The key computation in Eq. (3.5) is solution of the linear system for $\mathbf{C}$. As seen in Section 2.3, use of iterative solvers is much more efficient than direct solution. With the use of iterative solvers, accelerating the covariance matrix-vector product accelerates kriging as well.

We have already discussed the single-processor (IFGT[119, 72], Dual trees[51], FIGTREE[58]) and multiple-processor (GPUML [95, 101] accelerations for weighted Gaussian summations. Each of these approaches has its own advantages and disadvantages. Single-processor accelerations perform well for the low-dimensional data encountered in kriging, however, GPU-based acceleration outperform the single process acceleration for sample sizes upto $100,000$ [101]. To take advantage of both the ideas, we use an approximation approach on the GPU.

3.3.1 GPU parallelization

A naive parallelization of the covariance matrix-vector product on graphical processor [101] has been used previously to accelerate kriging [98]. GPUML [101]

was tuned for optimal performance across dimensions by employing the shared memory and registers to store covariance parameters and points x_i's. However, kriging involves low-dimensional data and a more specialized implementation would help here. *We use 2 specific rules to further improve on GPUML acceleration for the low-dimensional data.*

A Gaussian kernel (Eq. 3.4) decays as the distance between the two points increases. Therefore, **we avoid computing exponentials when their value are negligible, i.e. the distance between the points is greater than a threshold radius r.** We compute this r using the following rule:

$$\exp\left(\frac{-\|r\|^2}{h^2}\right) < \epsilon \rightarrow r > h\sqrt{-\ln \epsilon}, \tag{3.8}$$

where ϵ is the guaranteed error bound on the evaluated sum, and is set to 10^{-6} here.

In the data reconstruction problem, we encounter gridded data only and the distance between the data points are multiples of the grid-size along each dimension of interest (latitude and longitude). We therefore, **precompute the Gaussian kernels for multiples of the grid-size up to the cut-off radius r and store it on the texture memory**. Because the texture memory is cached, this not only avoids recomputations, but also enables a faster access to the stored exponential values.

Fig. 3.1 shows the computational improvement of our new approach over GPUML across various datasizes and Gaussian bandwidths h. We obtain up to 2X speedup over GPUML. The savings by our approach is maximal at lower and moderate bandwidths, because of a lower cut-off radius, and hence better cache

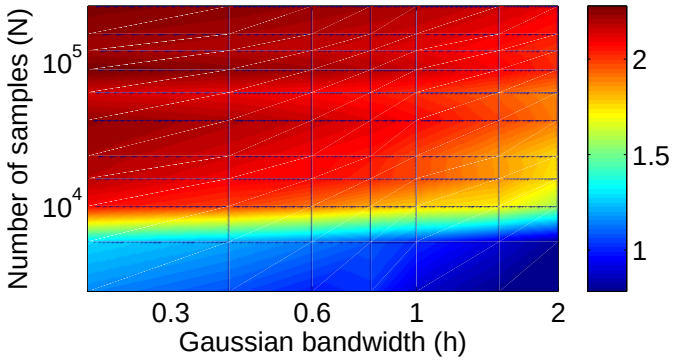


Figure 3.1: [color] *Speedup of our new implementation over the original GPUML [101]*

utilization. The speedup increases for larger data sizes because of the savings due to minimum Gaussian evaluations.

3.3.2 Performance comparison

In all experiments the host processor is an Intel Xeon Quad-Core 2.4GHz with 4GB RAM. The GPU is a Tesla C1060 which has 240 cores arranged as 30 multiprocessors. It has 4GB of global memory, 16384 registers per thread block and 16kB shared memory per multiprocessor.

We tested our kriging approach on synthetic data generated on a 2-D grid using the relation

$$f(x_1, x_2) = \sin(0.4x_1) + \cos(0.2x_2 + 0.2), \tag{3.9}$$

where $0.0 \leq x_1, x_2 \leq 10.0$. The surface represented by such a function is given in Fig. 3.2.

The data were removed from this 2-D grid to create artificial gaps, and the

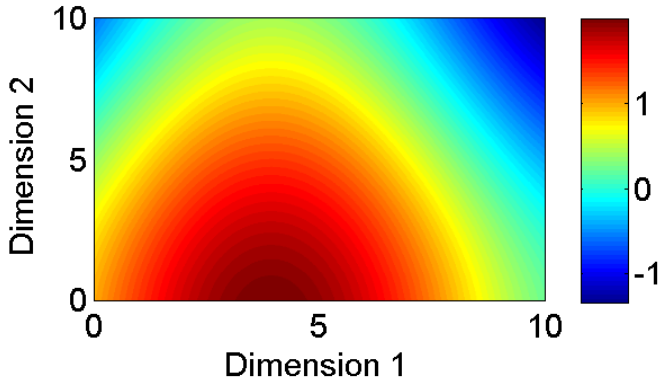


Figure 3.2: [color] *Synthetic surface generated and sampled to test the performance of our kriging*

gaps were then filled via kriging. The size of the gaps was varied to vary the problem size. We compared our approach against widely used kriging packages: Dace kriging [52] and mGstat kriging [38]. The comparison of the time taken by various kriging approaches is shown in Fig. 3.3. It can be seen that the proposed approach is computationally better than other approaches. Of course, the choice of kriging parameter is paramount for the reconstruction quality, but we defer discussion of this to Section 3.4.

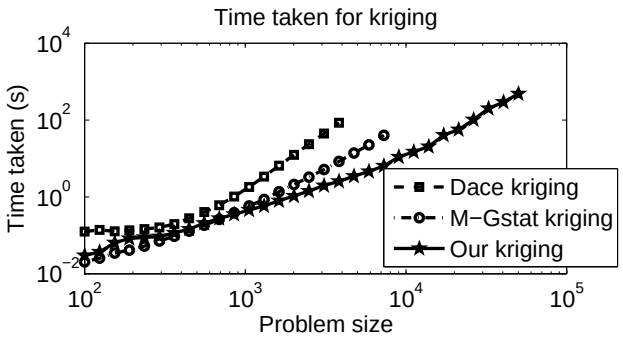


Figure 3.3: *Performance comparison across different kriging approaches for the synthetic data in Eq. (3.9).*

As mentioned earlier, one advantage of kriging is that it provides a way to estimate the variance of the reconstructed data. Fig. 3.4 shows the variance estimates for the reconstructed data, based a 3 sampling instances with 1%, 5% and 10% coverages respectively over the entire grid. It can be seen that when the amount of observed data is smaller, the reconstruction is more unreliable indicated by higher variance across the entire grid. When the coverage is reasonable (10% in Fig. 3.4), there is lower variance of the estimates and hence more reliability.

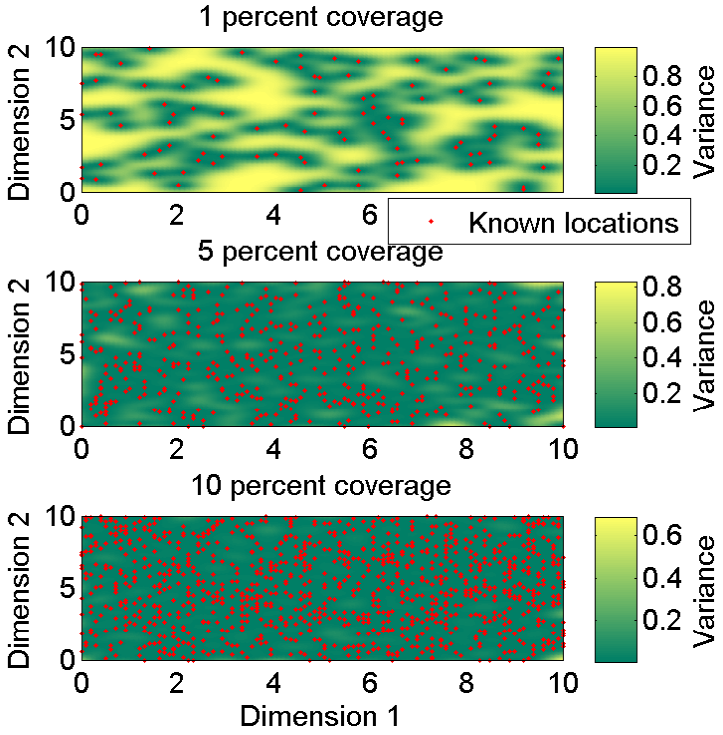


Figure 3.4: [color] *Variance estimates from kriging for reconstructing the gappy data in Eq. (3.9)*

3.4 Parameter estimation

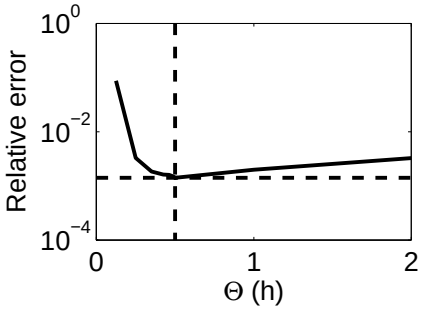
A key factor in the performance of kriging is the choice of the kriging parameters, Θ that give best reconstruction. For the Gaussian kernel, $\Theta = \{h\}$. We use modified gradient descent with cross-validation to estimate it.

The main assumption in this approach is that the error function is quadratic with respect to Θ. We divide our training data into *training* and *hold-off* sets. We train using the *training* set for various Θ values and test it on the *hold-off* set. The gradient descent algorithm is initialized with the Θ value that yields the least error. Different error metrics like the mean relative error, root mean square error, etc., can be used; we chose the mean relative error in our experiments. The parameter Θ is moved along the direction of negative gradient of the relative error evaluated using its finite-difference approximation. We repeat this for different *hold-off* sets and the median is chosen as the optimal value. This is summarized in Algorithm 1.

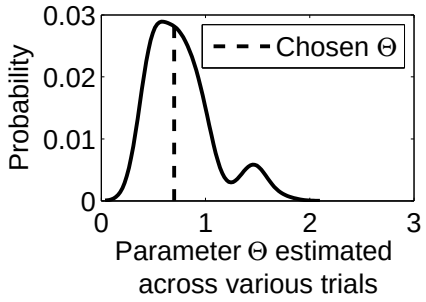
3.4.1 Validation

Our parameter estimation approach is validated with the $2-$D synthetic data (Fig. 3.2). The data is now divided into two parts, 90% for *training* and 10% for *testing*. A part of training data (10%) is set aside as hold-off data, and we estimate the parameter Θ based on the error on this hold-off. Fig. 3.5(a) shows the Θ relationship with the hold-off error and also the bandwidth chosen by our approach. The estimated Θ is used on the unseen data. This is repeated for various combinations of train, hold-off and test data. Fig.3.5(b) shows the values of Θ from

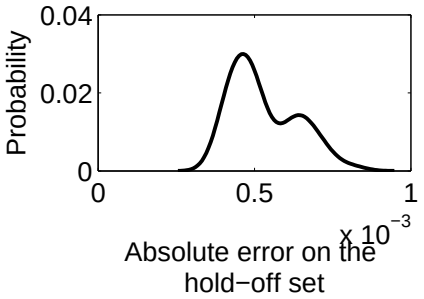
different trials, and the median chosen to be the optimal Θ. Figures. 3.5(c) and 3.5(d) show the corresponding hold-off and test error distributions for the chosen Θ in each trial. A similar test and hold-off error distribution validate the parameter estimation approach.



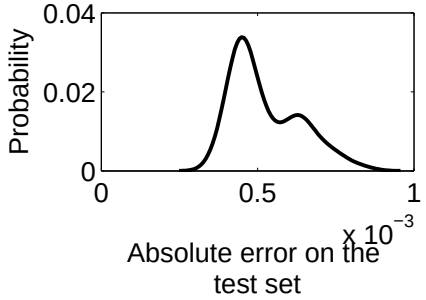
(a) *Parameter* Θ *vs the relative error. Dotted lines show* Θ *chosen by our approach and the corresponding error*



(b) *Statistics of the parameter* Θ *based on our estimation technique across* 100 *trials.*



(c) *Statistics of the error on the hold-off sets based on our estimation technique across* 100 *trials.*



(d) *Statistics of the error on the test sets based on our parameter estimation technique across* 100 *trials.*

Figure 3.5: *Statistical validation of the parameter estimation technique*

3.5 Experiments

Ocean color corresponds to chlorophyll concentrations, which is related to the energy at the bottom of the marine food chain and is a critical link to the carbon cycle [61]. Subtle changes in the ocean's color result from changes in the concentrations of marine phytoplankton, suspended sediments and dissolved substances in the water column. Knowing these is very much necessary for various industries, primarily fisheries and for proper modeling of the carbon and oxygen in the ocean[61]. We used the kriging framework to reconstruct the weekly chlorophyll concentration (ocean color) recorded for the Chesapeake bay and west coast sector of the Pacific ocean between 1998 and 2006 from the *Sea-viewing Wide Field-of-view Sensor (SeaWiFS)*.

Because of the spatio-temporal nature of these data, we modify the spatial kriging framework mentioned above to a spatio-temporal kriging. The parameters of this modified framework are $\Theta = \{h_s, h_t\}$, where h_s is the spatial Gaussian spread, h_t is the temporal Gaussian spread. Note that, we use the same h_s across both latitudes and longitudes, and hence our covariance function is isotropic. Using a different h for latitudinal and longitudinal scales (anisotropic covariance) can be appropriate in some cases but was not explored in this work. We estimate h_s first using Alg. 1 and then estimate h_t by fixing h_s. The 1998 chlorophyll data over both the regions was used to determine the parameters, which are then used for all the data from subsequent years.

3.5.1 SeaWiFS Chlorophyll Data

The SeaWiFS instrument [57] was designed to monitor the color of the global oceans and provides a time series of chlorophyll maps of the global ocean. Data from SeaWiFs provide insight into the understanding of the marine ecosystem and the ocean's role in the global carbon and other biogeochemical cycles, and have been utilized e.g. om [86].

We use the chlorophyll data from a SeaWiFS-equipped satellite and collected over the Chesapeake Bay and the west coast sector of the Pacific ocean. The Chesapeake bay (Fig. 3.6) data is collected from 78^oW to 73^oW and from 34^oN to 40^oN at a resolution of ~ 50 observations per degree, resulting in a grid of 240×288.. Similarly, the Pacific ocean (Fig. 3.7) data is collected from 159^oW to 100^oW and from 25^oN to 59^oN at a resolution of ~ 12 observations per degree, resulting in a grid of 600×400. The two areas offer datasets at different resolution, size and coverage enabling a good spectrum for our comparisons.

3.5.2 DINEOF

A widely used geostatistical reconstruction technique is based on Empirical Orthogonal Function (EOF) interpolation, and we used the data-interpolating-EOF (DINEOF) [1] for comparisons with kriging. DINEOF considers a spatio-temporal data as a $M \times N$ matrix X, M denoting the spatial space, and N denoting the temporal space. Performing a singular value decomposition (SVD) (otherwise known as the Empirical Orthogonal Functions / EOFs) on this matrix, $X = U\Sigma V$ results in

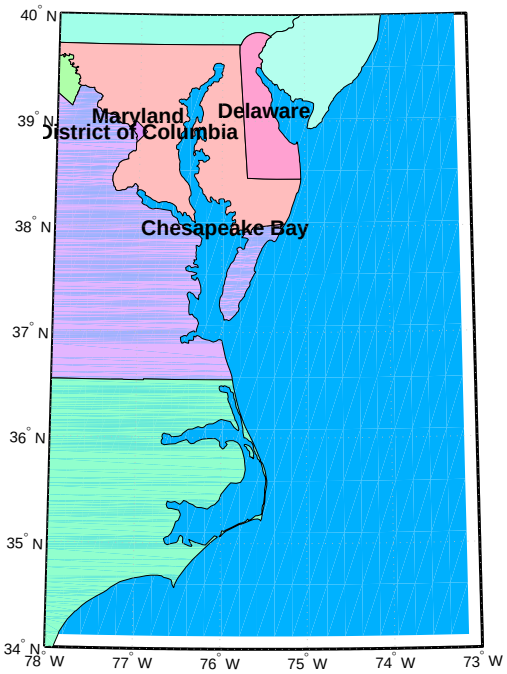


Figure 3.6: [color] *Kriging was used to reconstruct the data recorded over the Chesapeake Bay region shown here*

set of dominant singular values (EOF modes, Σ). EOF-based interpolation estimate the EOFs (U and V) and their modes (Σ) from the raw data filling the missing data of X with zeros, and reconstructs X using the dominant EOFs. This process is repeated to convergence and the final X matrix is the desired reconstruction. A detailed description is available in [1, 2, 4]. DINEOF does not account for the spatial location of a missing point or the spatial resolution of the data being reconstructed.

DINEOF software[1] uses the Lanczos method for the SVD, based on ARPACK [1] to reconstruct the gappy data, and is computationally efficient. Beckers et al. [4]

[1] `http://modb.oce.ulg.ac.be/mediawiki/index.php/DINEOF`

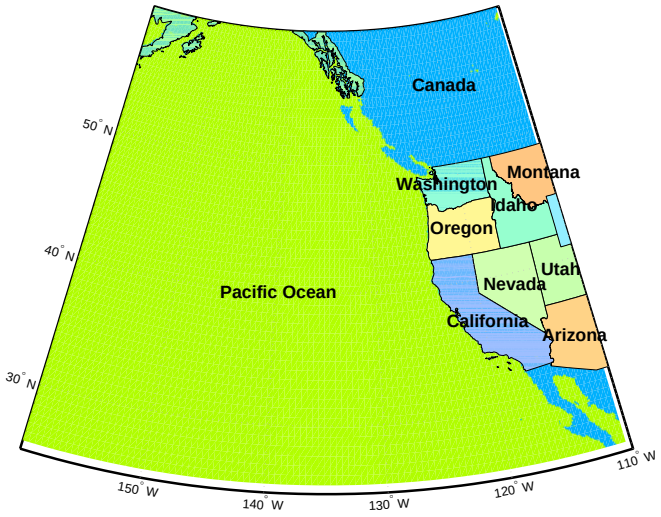


Figure 3.7: [color] *Kriging was used to reconstruct the data recorded over the Pacific ocean region shown here*

suggest that the performance of the reconstruction improves when the input data spans a period of few weeks within the required time-frame. DINEOF reconstruction takes 3 parameters, namely the *the number of EOF modes, size of the Lanczos subspace and temporal scale of the input data*. These parameters were set via cross-validation as prescribed in [1].

3.5.3 Reconstruction performance

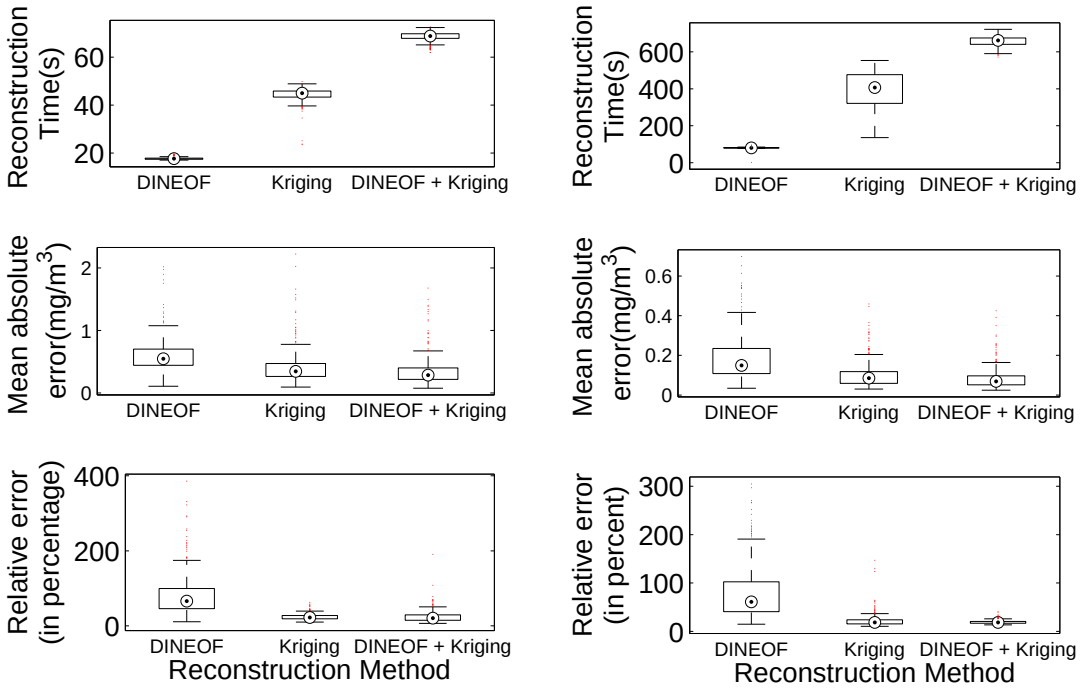
The computational complexity of DINEOF reconstruction (the primary bottleneck being SVD) is $\mathrm{O}(\min(MN^2, NM^2))$. On the other-hand, the computational complexity of our approach is $\mathrm{O}(k(NM)^2)$. Because only a few weeks of data are considered, the complexity is approximately $\mathrm{O}(N)$ for DINEOF and (GPU-accelerated)

$O(kN^2)$ for kriging. DINEOF and kriging based reconstruction are applied to both the regions above and the results are illustrated in Figs. 3.8(a) and 3.8(b). For the Chesapeake Bay, the average time taken by DINEOF is 17s, whereas kriging takes 44s. However, the average error (evaluated on different hold-off sets over multiple-trials) for DINEOF is $0.6mg/m^3$ (80% relative error) and for kriging is $0.4mg/m^3$ (24% relative error). Similarly, for the Pacific ocean, the average time taken by DINEOF is 80s, whereas kriging takes 390s. However, the average error for DINEOF is $0.2mg/m^3$ (79% relative error) and for kriging is $0.1mg/m^3$ (21% relative error). Despite the GPU acceleration, kriging's computational cost is expensive in comparison to DINEOF because of the corresponding quadratic and linear time complexities. However, there is a significant improvement in the reconstruction quality, as indicated by the relative and absolute errors.

Such errors are the subject of vigorous debate in the meteorological community, and we believe that the improved error and robustness of our method should be useful in applications. Since the debates on warming trends and uptake or release of CO_2 by the ocean and so on often depend on interpolating sparse data, accurate reconstructions of gappy data are often much more crucial than computational costs.

3.5.4 DINEOF-initialized kriging

EOF based approaches do not count the spatial-location of the data for reconstruction. Although this helps EOF methods to reliably reconstruct even sparsely-observed data, this results in a reduced impact of spatially-closer observations. On

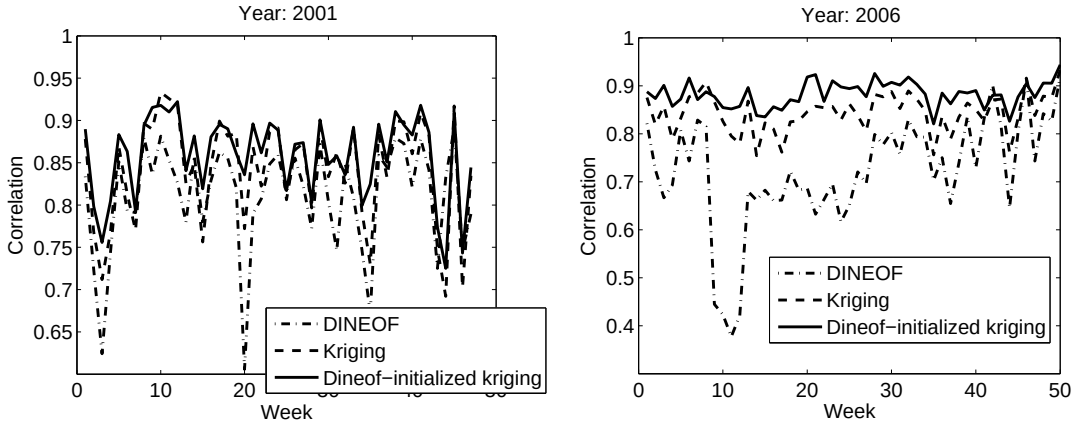


(a) Over the Chesapeake Bay (b) Over the Pacific Ocean

Figure 3.8: *Reconstruction performances*

the other hand, kriging performs very well locally, but fails when the data coverage is low (the variance of estimates is very high for lower coverage and hence less reliable as seen from Fig. 3.4). We therefore, combined both these techniques for the data reconstruction and the corresponding results are also shown in Figs. 3.8(a). The missing data are first reconstructed via DINEOF and followed by kriging to obtain the final estimate. Although this increases the computational cost of the overall approach (68s for Chesapeake Bay and 650s for Pacific ocean), the reconstruction quality improves over the reconstruction based on kriging alone. The mean error for Chesapeake Bay data is $0.34 mg/m^3$ (24% relative error) and for Pacific ocean is $0.08 mg/m^3$ (19% relative error).

An analysis on the correlation between the observed and reconstructed data reveals a better correlation for this DINEOF-initialized kriging over most of the analyzed time periods. Figs. 3.9(a) and 3.9(b) show the correlation plots for the Chesapeake Bay and Pacific ocean. The combined approach has the maximum correlation in most of the weeks, followed by the kriging based reconstruction. Weeks where DINEOF-initialized kriging has a low correlation, DINEOF and kriging based reconstruction also exhibit lower correlation, thus suggesting issues with observed data during these weeks.



(a) Over the Chesapeake Bay (b) Over the Pacific Ocean

Figure 3.9: *Correlation between the observed and reconstructed values*

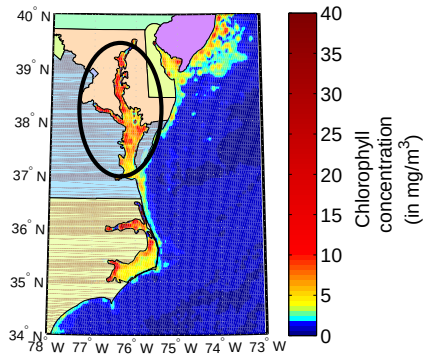
3.6 Conclusions

A kriging technique for statistically predicting missing data is formulated and its computational complexity is addressed via the use of graphical processors. The kriging parameters are estimated by a combination of standard cross-validation with an approximate gradient descent minimization. The resulting framework is used for

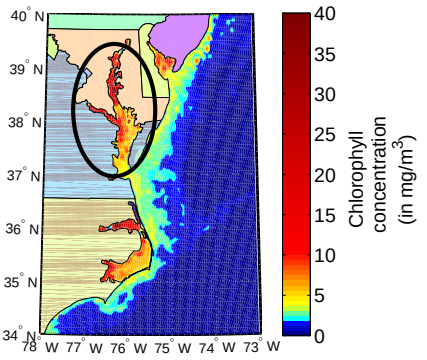
reconstructing the gappy data collected over the Chesapeake Bay and Pacific ocean, and the reconstruction quality was compared against an EOF-based reconstruction approach. While computationally kriging is expensive inspite of GPU acceleration because of its cubic complexity, it was seen that error residues are significantly lower compared to an EOF based approach. Finally, it was also observed that a combination of EOF and kriging yields better error residuals. The key algorithms and accelerations introduced in this paper will soon be release as an open source package.

Such reconstructed data find its application for initializing a meteorological model, data analysis and pattern discovery. For example, the effect of Hurricane Isabel[2] that hit the North Eastern United States between 6 and 20 of September 2003 has been well-studied. Roman et al. [74] observe an increased phytoplankton around the middle bay immediately after the hurricane, which later died down towards early October. As an after-effect, an increased zooplankton activity was also observed around mid-November. The kriged reconstructions during this period also exhibit these patterns on the chlorophyll as seen in Fig. 3.10, and our approach can be helpful for similar analysis in these directions.

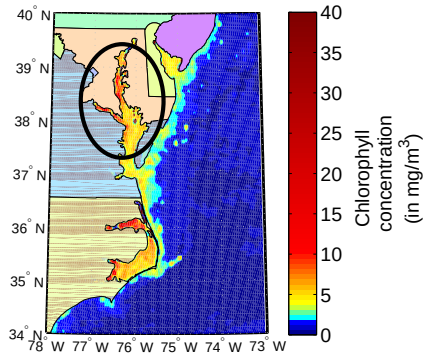
[2] http://en.wikipedia.org/wiki/Hurricane_Isabel



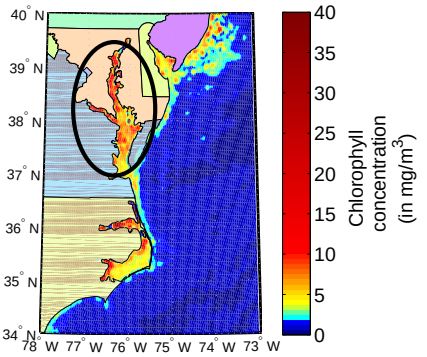
(a) Before ISABEL



(b) During ISABEL: Growth of phytoplanktons



(c) After ISABEL in early October



(d) After ISABEL in mid-November (Zooplanktons growth)

Figure 3.10: [color] *Chesapeake Bay before and after the* 2003 *Isabel hurricane*

Algorithm 1 Modified gradient descent for kriging parameter estimation

Given: Training locations X_{train} and corresponding values v_{train}

Given: Initial parameter value Θ_{init}

repeat

  Choose: Hold-off locations $X_{hold-off}$ and corresponding values $v_{hold-off}$

  Initialize step-size α, decay-factor β

  Evaluate $v^{*-1}_{hold-off}$ and $v^{*0}_{hold-off}$using x_{train} and y_{train} for Θ_{init} and $\beta\Theta_{init}$

  Define $err^{*j} = (v^{*j}_{hold-off} - v_{hold-off})/v_{hold-off}$ and evaluate err^{*-1} and err^{*0}.

  $iteration = 0$

  repeat

    $iteration++$

    Evaluate $\Delta_\Theta err^{*iter}$ via finite-differences

    Move Θ_{iter} along the negative of this gradient direction weighted by α

  until convergence of Θ

  $\Theta_{set} = \Theta_{last}$

until Required number of trials reached

Θ_{final}=median(Θ_{sets})

return Θ_{final}

Chapter 4

Preconditioned Krylov solvers for kernel regression

It was seen in Chapter 2 that the core algorithm in several kernel machines involves a number of linear algebra operations on matrices of kernel functions, which take as arguments the training and/or the testing data. A kernel function $\Phi(x_i, x_j)$ generalizes the notion of the similarity between data points. Given $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}, x_i \in R^d$, the kernel matrix entries are given by $\Phi(x_i, x_j)$,

$$\hat{\mathbf{K}} = \begin{pmatrix} \Phi(x_1, x_1) & \ldots & \Phi(x_1, x_N) \\ \vdots & \ddots & \vdots \\ \Phi(x_N, x_1) & \ldots & \Phi(x_N, x_N) \end{pmatrix}. \tag{4.1}$$

Φ is generally chosen to reflect prior information about the problem. In the absence of any prior knowledge, the Gaussian is the most widely used kernel. Most kernel methods use the kernel matrix in regularized form,

$$\mathbf{K} = \hat{\mathbf{K}} + \gamma \mathbf{I}; \tag{4.2}$$

with γ chosen appropriately according to the problem.

Kernel regression is one of the popular kernel methods that appears in many variations. We had seen two variants in Gaussian process regression[69] (Chapter 2) and geostatistical kriging[42] (Chapter 3). While the formulations in each of these differ slightly, the key computation in all these requires the solution of a linear system with $\mathbf{K}$. Direct solution for a dense kernel matrix system has a time

complexity $O(N^3)$ and memory complexity $O(N^2)$, which prevents its use with large datasets. Therefore, iterative Krylov methods [80] are used to address this partially by reducing the time complexity to $O(kN^2)$, k being the number of iterations [34, 21]. The dominant cost per Krylov iteration is a kernel matrix-vector product (MVP), which has been accelerated on a single core and multiple cores (Chapter 2). *In these fast kernel MVP, there is a trade-off between accuracy and speed, and usually a MVP of reduced accuracy can be obtained faster.* The fast MVPs scale well for large datasets, however the convergence rate for large problems suffers since the matrix might not always be "well-conditioned". To speedup the iterative methods for large scale kernel regression, apart from using fast MVP, we need to reduce the number of iterations and where possible use lower accuracy MVP without affecting the solution accuracy. We address this problem in this chapter by introducing fast preconditioners for a flexible Krylov solver.

For symmetric marices, the convergence of the Krylov methods is determined by the conditioning of the kernel matrix characterized by the matrix condition number κ ($\kappa \geq 1$),

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}}, 1 \leq \kappa < \infty. \tag{4.3}$$

where $\lambda_{\max}$ is its largest eigenvalue and $\lambda_{\min}$ the smallest eigenvalue. For smaller κ, the Krylov methods converge faster. When a kernel matrix is ill-conditioned, there is a significant decrease in rate of convergence, necessitating a "preconditioner" [80] to improve the conditioning of the matrix used in iteration and accelerate convergence. Preconditioning has been suggested for kernel matrix computations [21, 60], but to

our knowledge, there has been *no previous work on a method to actually design a preconditioner* for such matrices.

To be effective, the preconditioner matrix construction cost should be small, and it should be able to take advantage of fast matrix vector products. We propose a novel preconditioner that improves convergence and has the added benefit that it utilizes the matrix-vector product acceleration available for the kernel matrix.

The chapter is organized as follows. We introduce Krylov methods and their convergence properties in Sec. 4.1 and survey different preconditioning techniques in Sec. 4.2. The new preconditioner is introduced and its parameters and convergence are studied in Sec. 4.3. Finally we test its performance on synthetic and standard datasets in Sec. 4.4.

4.1 Krylov methods

Krylov solvers are the methods of choice for many linear algebra problems (solving linear systems, eigenvalue problems, optimization). They iteratively improve the solution of a "cost-minimization" problem over a set of basis vectors (the Krylov basis) created via matrix vector products of the matrix under consideration. While a detailed discussion and analysis can be found in [80, 78], we provide a brief overview here in the context of solving linear systems.

Consider the solution of the linear system, $\mathbf{K}x = b$. Krylov methods begin with an initial guess $x^{(0)}$ and minimize the cost function $r^{(k)} = b - \mathbf{K}x^{(k)}$ in some norm, by moving the iterates along directions in the Krylov subspace

$\mathcal{K}_k = span(b, \mathbf{K}b, \ldots, \mathbf{K}^{k-1}b)$. The directions are augmented over each Krylov iteration, a significant difference from simpler iterative approaches like Gauss-Siedel where the next iterate depends only on the previous one.

At the k^{th} iteration of the Krylov methods, an orthogonal matrix $V^{(k)} = [v_1, v_2, \ldots, v_k]$ is generated such that columns of $V^{(k)}$ span the Krylov subspace $\mathcal{K}_k$. This would result in an "Arnoldi computation" [80],

$$\mathbf{K}V^{(k)} = V^{(k+1)}\bar{\mathbf{H}}^{(k)}, \tag{4.4}$$

where $\bar{\mathbf{H}}^{(k)}$ is an augmented Hessenberg matrix,

$$\bar{\mathbf{H}}^{(k)} = \begin{pmatrix} h_{1,1} & h_{1,2} & h_{1,3} & \ldots & h_{1,k} \\ h_{2,1} & h_{2,2} & h_{2,3} & \ldots & h_{2,k} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \ldots & 0 & h_{k,k-1} & h_{k,k} \\ 0 & \ldots & 0 & 0 & h_{k+1,k} \end{pmatrix},$$

where $h_{i,j} = (v_j^T \mathbf{K} v_i)$. The next iterate $x^{(k)}$ is then obtained by solving the least squares problem, $\min_y \|\bar{\mathbf{H}}^{(k)}y - \beta e_1\|$, where $e_1 = [1, 0, \ldots, 0]^T$, and the iterate is given by,

$$x^{(k)} = V^{(k)}y. \tag{4.5}$$

This is the *Arnoldi* iteration for system solution [80].

The *conjugate gradient (CG)* method is the most widely used Krylov method with symmetric matrices. For symmetric $\mathbf{K}$, $\bar{\mathbf{H}}^{(k)}$ in Eq. 4.4 is tridiagonal leading to the particularly efficient CG method. The *generalized minimum residual (GMRES)*

is usually used for non-symmetric problems though it can be used in the symmetric case as well. GMRES directly extends the Arnoldi iterations to minimize the residuals $r^{(k)}$ in the $2-$norm. CG minimizes the $\mathbf{K}$-norm of the residual and utilizes the conjugacy in the resulting formulation, which results in not having to store the Krylov basis vectors. CG, therefore, results in a more efficient (lower cost per iteration) specialized formulation than the GMRES. The specific algorithmic differences are detailed in [80]. Kernel matrices are symmetric and satisfy the Mércer conditions $a^T\mathbf{K}a > 0$, for any a; and hence $\mathbf{K}$ is positive definite. Therefore, CG has been the preferred choice for kernel matrices [34]; however, GMRES has also been used [21].

4.1.1 Fast matrix-vector products:

The key computation in each Krylov step is the matrix-vector product, $\mathbf{f} = \mathbf{Kq}$ or $f(x_j) = \sum_{i=1}^{N} q_i \Phi(x_i, x_j)$. We used GPUML from Chapter 2 to parallelize kernel summation on graphical processors (GPUs).

4.1.2 Convergence of Krylov methods:

The convergence rate of iterative approaches is given by the ratio of the error (e_k) at k^{th} iteration to the initial error (e_0) in some norm. For example, the ratio for CG [80] is,

$$\frac{\|e_k\|_{\mathbf{K}}}{\|e_0\|_{\mathbf{K}}} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k. \tag{4.6}$$

A more complicated expression may be derived for GMRES [80].

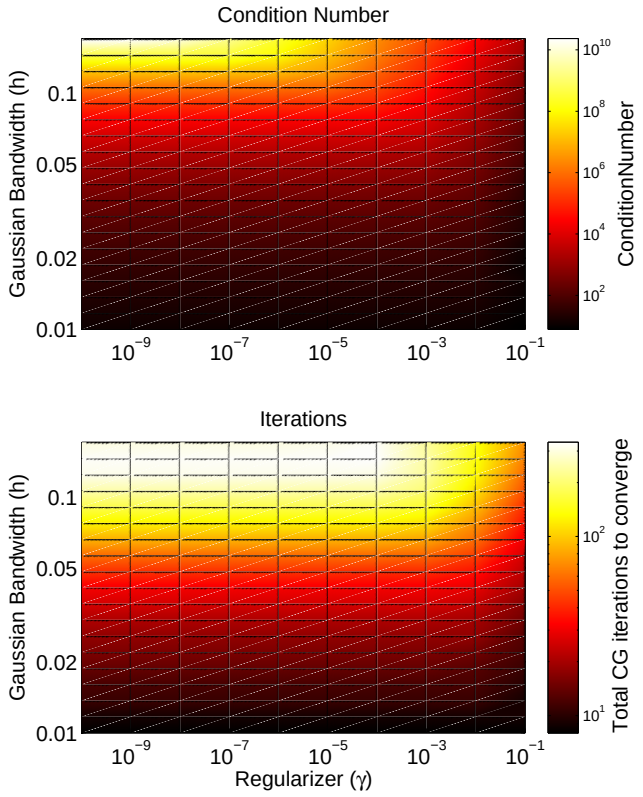


Figure 4.1: [color] *Effect of kernel hyper-parameters on the matrix conditioning and CG iterations*

4.1.3 Need for preconditioning:

The condition number of kernel matrices depends on the data point distribution and the kernel hyper-parameters. For the Gaussian (Eq. 2.2), the hyper-parameters are the bandwidth h and the regularizer γ. While x_i's are given, the hyper-parameters are generally evaluated using maximum-likelihood. Fig. 4.1 shows the condition number and number of CG iterations to converge for a kernel matrix constructed from data points generated uniformly at random inside a unit cube. There is a direct correspondence between the condition number and number of CG

iterations. For larger regularizer and smaller bandwidths, the convergence is much better. The data point distribution influences the conditioning as well. It is however not possible to hand select these parameters for each problem. It is therefore necessary to "precondition" [80] the system to be solved to yield better Krylov convergence irrespective of the underlying matrix conditioning.

4.2 Preconditioning techniques

Consider $\mathbf{K}x = b$. A left preconditioner ($\mathbf{M}^{-1}$) operate on this system as,

$$\mathbf{M}^{-1}\mathbf{K}x = \mathbf{M}^{-\mathbf{1}}b; \tag{4.7}$$

and a right preconditioner operates as,

$$\mathbf{K}\mathbf{M}^{-1}y = b, \quad y = \mathbf{M}x. \tag{4.8}$$

The preconditioner $\mathbf{M}^{-1}$ should be chosen such that the resulting matrices ($\mathbf{M}^{-1}\mathbf{K}$ or $\mathbf{K}\mathbf{M}^{-1}$) have a low condition number. An ideal preconditioner ($\mathbf{M}^{-1}$) is a matrix that well approximates $\mathbf{K}^{-1}$, but is easy to compute.

4.2.1 Conventional preconditioners

Standard preconditioners used in the literature are for sparse matrices that arise in the solution of differential equations, and include those based on Jacobi and Symmetric Successive Over-Relaxation (SSOR) algorithms. For general sparse matrices, incomplete LU preconditioners are often used. The triangular factors $\mathbf{L}$ and $\mathbf{U}$ for a sparse matrix may not be sparse, but *incomplete LU* factorizations leads

to sparse $\mathbf{L}$ and $\mathbf{U}$ matrices by forcing the coefficients leading to zero entries of the sparse matrix to zero. For a dense matrix, elements are sparsified using a cut-off threshold.

Preconditioners to radial basis function interpolation are a closely related problem and fast preconditioners have been proposed in this direction [3, 29, 37]. However, these approaches are limited to low dimensions (~ 3) and are computationally inefficient for larger dimensions.

4.2.2 Flexible preconditioners

As seen from Eqs. (4.7) and (4.8), a left preconditioner modifies the right-hand side in the problem whereas the right preconditioner leaves it as is. This property of right preconditioners can be exploited to create "flexible" preconditioning techniques where a different preconditioner can be used in each Krylov step [79, 89, 62], since the preconditioner only appears implicitly. Flexible preconditioning can be used with both CG [62] and GMRES [79].

Although many papers have shown the convergence of flexible preconditioners under exact arithmetic, it is very hard to estimate the convergence rate or the number of outer iterations accurately under inexact arithmetic since the underlying subspaces, $x_0 + span\{\mathbf{M_1}^{-1}v_1, \mathbf{M_2}^{-1}v_2, \ldots, \mathbf{M_k}^{-1}v_k\}$ are no longer a standard Krylov subspace. This affects CG since the new subspace impacts the underlying conjugacy. Notay [62] proposes 2 modifications to a preconditioned flexible CG. The iterates should be "reorthogonalized" at each step to maintain conjugacy; and the

preconditioner system should be solved with high accuracy. Flexible preconditioners are however easily used with GMRES. This fact will be observed in results below, where a poorer performance is observed for flexible CG relative to flexible GMRES.

The algorithmic details of flexible GMRES is enlisted in Algorithm 2, and the corresponding unpreconditioned version is obtained by replacing the Ms in Algorithm 2 with identity matrices. The iterations are stopped when $\epsilon = \frac{b-\mathbf{K}x_i}{N}$ goes below a certain tolerance. Similar extension is available for CG as well and is shown in Algorithm 3. The stopping criterion is similar to the flexible GMRES. The unpreconditioned CG is obtained by removing the reorthogonolization step in flexible CG and replacing M with an identity matrix.

4.2.3 Krylov method as a flexible preconditioner

In Algorithms 2 and 3, all that is needed to prescribe the right preconditioner is a black-box routine which returns the solution to a linear system with the preconditioner matrix $\mathbf{M}$. Instead of explicitly specifying $\mathbf{M}^{-1}$, it is possible to specify it implicitly by solve a linear system with $\mathbf{M}$ using another Krylov method such as CG. *However, because this iteration does not converge exactly the same way each time it is applied, the use of an iterative method as preconditioner is equivalent to using a different* $\mathbf{M}$ *for each iteration [89] in exact arithmetic.* We refer to the preconditioner, operating with matrix $\mathbf{M}$ as "inner Krylov" and to the main solver operating on $\mathbf{K}\mathbf{M}^{-1}$ as "outer Krylov".

Algorithm 2 Flexible GMRES [79]

1: $r_0 = (b - \mathbf{K}x_0)$, $\beta = \|r_0\|_2$ and $v_1 = r_0/\beta$

2: Define the $m + 1 \times m$ matrix, $\bar{H}_m = \{h_{i,j}\}_{1 \leq i \leq j+1; 1 \leq j \leq m}$

3: **for** $j = 0$ to m **do**

4: Solve $\mathbf{M_j} z_j = v_j$ (***inner preconditioner***)

5: $w = \mathbf{K} z_j$ (matrix-vector product)

6: **for** $i = 0$ to j **do**

7: $h_{i,j} = (w, v_i)$, $w = w - h_{i,j} v_i$

8: **end for**

9: $h_{j+1,j} = \|w\|_2$, $v_{j+1} = w/h_{j+1,j}$

10: **end for**

11: $\mathbf{Z}_m = [z_1, \ldots, z_m]$,

12: $y_m = \arg\min_y \|\beta e_1 - \bar{H}_m y\|_2$, $x_m = x_0 + \mathbf{Z}_m y_m$

13: IF satisfied STOP, else $x_0 = x_m$ and GOTO 1

4.3 Preconditioner for kernel matrices

Conventional preconditioners require construction of the complete matrix initially, followed by expensive matrix decompositions. Thus they have a computational cost of $\mathrm{O}(N^3)$ and a memory requirement of at least $O(N^2)$. Additionally, the preconditioner evaluations will require a $\mathrm{O}(N^2)$ "unstructured" matrix-vector product, which does not have any standard acceleration technique and is hard to parallelize. This limits application to very large datasets and will ruin any advantage gained by the use of fast matrix-vector products (as will be seen later in Sec.

Algorithm 3 Flexible Conjugate Gradient [62]

1: $r_0 = (b - \mathbf{K}x_0)$, $i = 0$

2: **while** r_i is not sufficient small **do**

3: Solve $\mathbf{M_i} z_i = r_i$

4: $d_i = z_i - \sum_{k=0}^{i-1} \frac{(z_i, \mathbf{K}d_k)}{(d_k, \mathbf{K}d_k)} d_k$ [Reorthogonolization]

5: $x_{i+1} = x_i + \frac{(d_i, r_i)}{(d_i, \mathbf{K}d_i)} d_i$

6: $r_{i+1} = r_i - \frac{(d_i, r_i)}{(d_i, \mathbf{K}d_i)} \mathbf{K}d_i$

7: $i = i + 1$

8: **end while**

4.3.5).

This leads us to consider to the key requirement for any preconditioning approach for a kernel matrix: ***the preconditioner should operate with an asymptotic time complexity and memory requirement that are at least the same as the fast matrix vector product.*** Otherwise the application of the preconditioner would increase the cost. This leads us to one of the main contributions of this chapter – a particularly simple construction of a right preconditioner, which also has a fast matrix vector product.

We propose to use a *regularized kernel matrix* $\mathbf{K}$ as a right preconditioner,

$$\mathbf{M} = \mathbf{K} + \sigma \mathbf{I}. \tag{4.9}$$

Regularization is a central theme in statistics and machine learning [111], and is not a new concept for kernel machines, e.g. ridge regression, where the kernel matrix ($\hat{\mathbf{K}}$) is regularized as $\hat{\mathbf{K}} + \gamma \mathbf{I}$. However, the γ is chosen by statistical techniques,

and hence cannot be controlled.

Our use of this old trick of regularization is in a new context – in the preconditioner matrix $\mathbf{M}$. The simple prescription achieves the following properties:

- improves the condition number of the matrix $\mathbf{M}$, leading to faster convergence of the inner iterations
- improves condition number of the outer matrix $\mathbf{K}\mathbf{M}^{-1}$.

To translate this idea in to a useful preconditioner, we need a prescription for selecting the regularization parameter σ and specifying the accuracy ϵ to which the inner system needs to be solved. We use CG to solve the inner preconditioner system.

It is important to note here that Krylov methods are numerical techniques whose formulation rely on exact arithmetics; therefore performances with matrices $\mathbf{K}$ and $\mathbf{K} + \sigma\mathbf{I}$ are significantly different, as was also seen in their convergence and conditioning in Fig. 4.1.

4.3.1 Preconditioner acceleration

A preconditioner improves the convergence of the iterative approach at the expense of an increased cost per iteration (cost associated with the preconditioner construction amortized over all iterations and cost of applying the preconditioner matrix). For a preconditioner to be useful, the total time taken by the preconditioned approach should be less than the unpreconditioned approach.

The key advantages of the proposed preconditioner is that, because $\mathbf{M}$ is derived from $\mathbf{K}$, given $\mathbf{X} = \{x_1, x_2, \ldots, x_N\}, x_i \in R^d$ it is not necessary to explicitly

construct the preconditioner $\mathbf{M}^{-1}$. Further, the key computation in the inner Krylov iteration is a matrix-vector product, $\mathbf{M}x$. This can be accelerated using the same fast algorithm as for $\mathbf{K}$. Further, the preconditioner system only needs to be solved approximately (with a low residual CG tolerance [accuracy to which the CG is solved] and with a lower accuracy matrix-vector product). In the experiments we use low-accuracy fast matrix vector products for the inner iterations (single precision on the GPU). For the outer iterations, the products are performed in double-precision (double-precision on GPUs are slower).

4.3.2 Preconditioner parameters

To guarantee an efficient preconditioner, it must be ensured that the CG used for the preconditioner has rapid convergence. While the data points and kernel hyper-parameters cannot be controlled, the preconditioner regularizer σ can be. The convergence of the CG for a kernel matrix for different σ's is shown in Fig. 4.1. It can be seen that for large enough σ, the CG converges rapidly. The CG can also be forced to have an early termination by setting a low solution accuracy (ϵ).

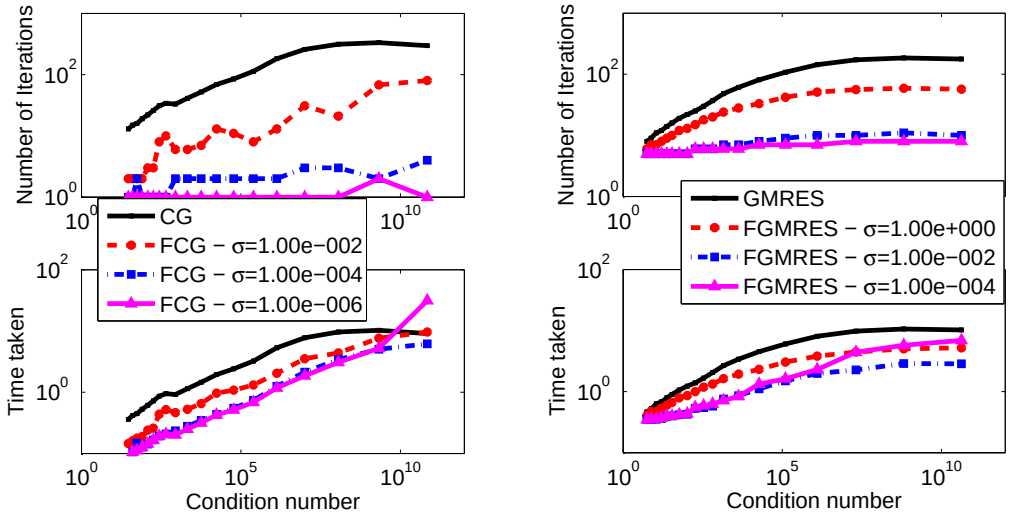
4.3.3 Effect of regularization parameter (σ):

In flexible Krylov methods, the outer iteration solves $\mathbf{K}\mathbf{M}^{-1}y = b$, and the inner CG solves $\mathbf{M}x = y$. For small values of σ, the preconditioner $\mathbf{M}$ is closer to the actual matrix $\mathbf{K}$. Therefore, the outer GMRES solves a better conditioned system; however, when $\mathbf{K}$ is ill-conditioned, $\mathbf{M}$ will also be somewhat ill-conditioned, thus

slowing the convergence of the inner iterations.

To demonstrate this, we generated data as before by taking 2000 random samples in a unit cube and generated a matrix for the Gaussian kernel. We tested the convergence with this preconditioner for various regularizer values (Figs. 4.2(a) and 4.2(b)). For smaller σ, the convergence of the outer Krylov iterations is faster, but the cost per iteration increases due to slow convergence of the inner Krylov iterations. Large regularization results in a poor preconditioner $\mathbf{M}$. A moderate value of the regularizer would therefore be an appropriate choice. This is consistently observed in both FCG and FGMRES, however, because of its formulation, the optimal FCG regularizer σ is an order of magnitude lower than that for FGMRES.

The choice of a regularizer involves a trade-off between the preconditioner's accurate representation of the kernel matrix and its desired conditioning.



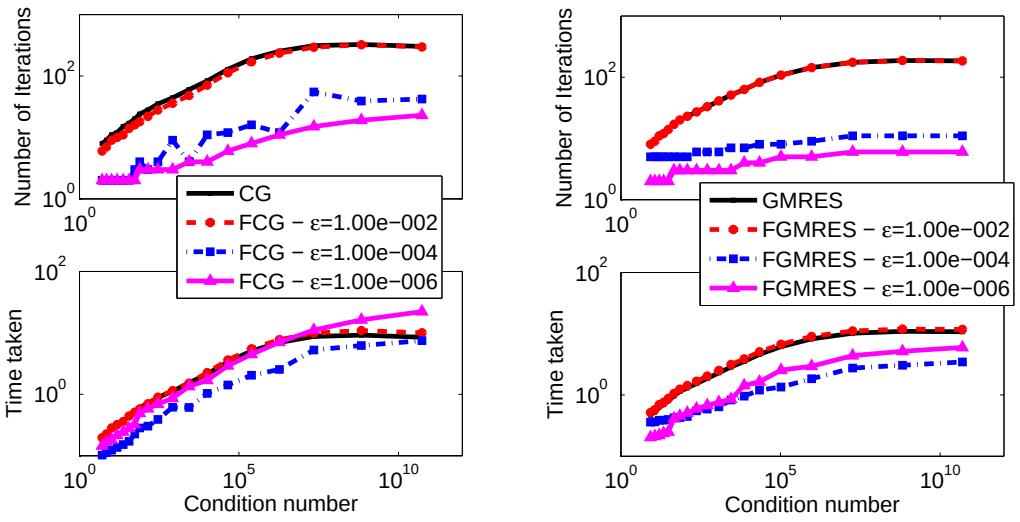
(a) Effect of regularizer σ on flexible CG (b) Effect of regularizer σ on flexible GMRES

Figure 4.2: [color] *Effect of regularizer σ on the convergence for FCG and FGMRES.*

4.3.4 Effect of CG tolerance (ϵ):

We tested the performance of the preconditioner for various tolerances in the inner iterations (Figs. 4.3(a) and 4.3(b)). There is a consistent improvement in the outer Krylov convergence for more precise convergence settings of the inner Krylov solver. However, the cost of inner iterations increases if this is set too fine. Therefore, a moderate value of ϵ works best for both FCG and FGMRES in terms of computational costs.

The choice of tolerance for CG iterations is a trade-off between the required solution accuracy of the preconditioner system (and hence the convergence of the outer iterations) and the related computational cost.



(a) Effect of inner CG tolerance ϵ on flexible CG

(b) Effect of inner CG tolerance ϵ on flexible GMRES

Figure 4.3: [color] *Effect of CG tolerance ϵ on the convergence for FCG and FGMRES.*

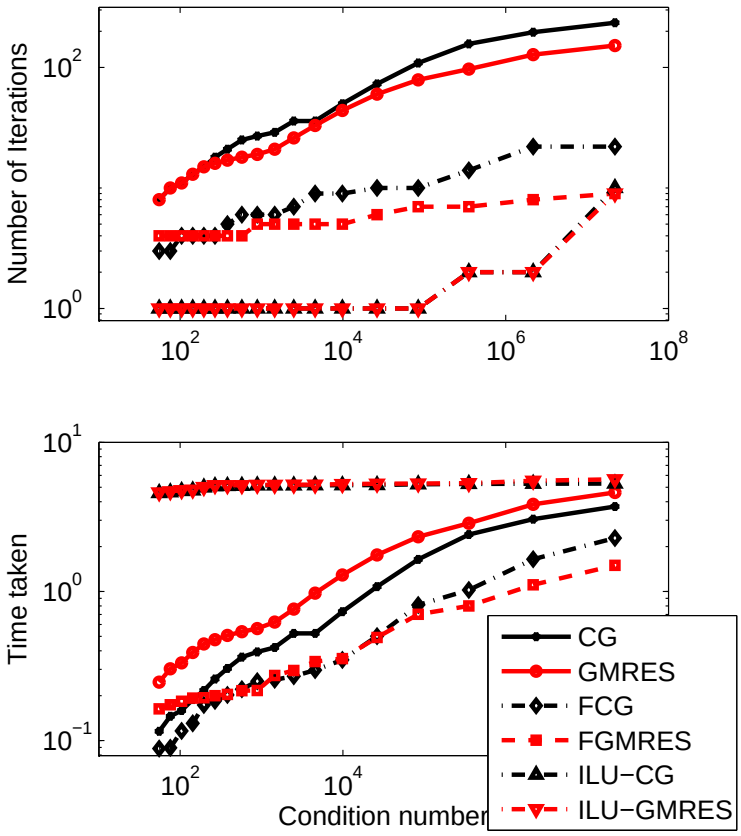


Figure 4.4: [color] *Performance of the proposed preconditioner with CG and GMRES against ILU-preconditioned and unpreconditioned versions*

4.3.5 Test of convergence

We compared the performance of flexible CG and flexible GMRES against ILU preconditioned CG and GMRES and the unpreconditioned CG and GMRES.

We set the preconditioner σ and tolerance ϵ to $\{10^{-4}, 10^{-4}\}$ respectively for FCG and $\{10^{-2}, 10^{-4}\}$ for FGMRES respectively. 2000 data points were generated randomly in a unit cube for testing the convergence. The computational performance and convergence is shown in Fig. 4.4. The number of iterations of the pre-

conditioned approaches are always less than those for the unpreconditioned cases. The computational cost per iteration is the least for CG compared to GMRES, FCG, and FGMRES. ILU based preconditioners are marginally better in convergence (iterations) compared to our approach for better conditioned cases. But ILU (and other similar preconditioners) require explicit kernel matrix construction and rely of sparsity and the absence of these properties in kernel matrices result in significantly higher computational cost compared to our preconditioners as well the unpreconditioned solver. This make conventional preconditioners impractical to be used with large datasets.

We see from the experiments above that FCG needs increased accuracy of the inner linear system solution. In contrast, FGMRES is more forgiving of inner linear system error and only requires coarse accuracy to reduce the number of outer iteration to the same magnitude as FCG. On the other hand, especially for the ill-conditioned matrices, solving the inner Krylov method with fine accuracy takes much more time. Hence, given the ill-conditioned kernel matrices, the best FMGRES has the smaller number of outer iterations as well as smaller total computation time.

The unpreconditioned algorithm of choice is CG, because of its lower storage and comparative efficiency. However, FGMRES is the method of choice for preconditioned iterations. Note that while GMRES requires extra storage in comparison to CG, FCG also requires this extra storage (for reorthogonalization), and we do not pay a storage penalty for the choice of FGMRES over FCG. In the rest of the chapter, we accordingly use FGMRES.

4.4 Experiments

The performance of the preconditioner is illustrated on various datasets on different variants of kernel regression. We first look at Gaussian process regression with a Gaussian kernel and then extend the preconditioned approach to a generalized (non-Gaussian) kernel. We finally experiment on kriging [42] and report results on a large geostatistical dataset.

Although dataset-specific tuning of the preconditioner parameters can yield better results, this is impractical. We therefore use the following rules to set the preconditioner parameters.

- The tolerance (ϵ) for the preconditioner system solution is set at an order of magnitude larger than the outer iteration tolerance (e.g., if the outer tolerance was 10^{-4}, the inner tolerance was set to 10^{-3}).

- Similarly, the preconditioner regularizer σ is also set to an order of magnitude higher than the kernel regularizer γ. When the outer regularizer is 0, the inner regularizer is set to 10^{-3}

While this might not yield the best preconditioner system, it performs well in most cases from the experiments. In all the experiments, the outer iteration tolerance was set to 10^{-6}.

4.4.1 Gaussian process regression (GPR)

Mackay et al. [34] suggest using *CG* to solve the GPR. Table 4.1 shows a comparison of the performance of Gaussian process regression based on our precon-

ditioner and our implementation of the CG approach in [34] on various standard datasets[1]. The matrix vector product associated with the CG approach in [34] was also accelerated using GPUML.

The convergence of the FGMRES is consistently better than the unpreconditioned approach. Although for smaller datasets the computational performance of the preconditioned and unpreconditioned [34] solvers are comparable, the performance of FGMRES gets better for larger data sizes. This is because, for larger problems, cost per iteration in both CG and FGMRES increases, and thus a FGMRES which converges faster becomes significantly better than the CG-based approach.

Low rank approaches [92, 83, 91] also address the time complexity in kernel regression by working on an "active set" of set M and reducing the time to $\mathrm{O}(M^2N)$. We compared with the low rank Gaussian processes based on [91], and found our approach to be superior. Because these approaches involve the solution of a large optimization problem, straightforward algorithmic acceleration or parallelization is not possible. Since the methods and accelerations used in this chapter are significantly different from those in [91], we have not reported these here, and defer a detailed comparison to the future.

To illustrate the applicability of our preconditioner to non-Gaussian kernels, we tested it on the Matern kernel[114],

$$k(x_i, x_j) = (1 + \sqrt{3}d_{ij}) \exp(-\sqrt{3}d_{ij}), \tag{4.10}$$

where $d_{ij} = \sqrt{\frac{\|x_i - x_j\|^2}{h^2}}$. We used the GPR framework for a binary classification

[1] `www.liaad.up.pt/~ltorgo/Regression/`

Datasets	GM [34]	FGMRES	Datasets	GM [34]	FGMRES
Robot-arm	23.81s	**11.79**s	*Bank*	49.40s	**37.74**s
(9×8192)	(75)	(**4**)	(33×4500)	(38)	(**3**)
Census	117.45s	**90.31**s	*Ailerons*	131.34s	**128.22**s
(9×22784)	(42)	(**4**)	(41×7154)	(31)	(**4**)
Census (2)	663.70s	**482.50**s	*Sarcos*	1399s	**797.9***s*
(17×22784)	(83)	(**5**)	(28×44484)	(50)	(**4**)

Table 4.1: *Performance of our FGMRES based Gaussian process regression against the CG based approach by Gibbs and Mackay (GM) in [34]; d is the dimension and N is the size of the dataset with the Gaussian kernel. Total time taken for prediction is shown here, with the number of iterations for convergence indicated within parenthesis. The mean error in prediction between the two approaches was less than* 10^{-6} *in all the cases.*

problem and tested it on several standard datasets[2]. The results are tabulated in Table 4.2. Here again, FGMRES has a better computational performance than the CG solver, thus illustrating its validity on non-Gaussian kernels.

4.4.2 Kriging

We compared the FGMRES-based kriging against the CG versions on the ocean chlorophyll concentration data recorded along the Pacific coast of North Amer-

[2] `www.csie.ntu.edu.tw/~cjlin/libsvmtools/`

Datasets	GM [34]	FGMRES	Datasets	GM [34]	FGMRES
Glass	**0.32**s	0.36s	*German*	**0.33**s	0.45s
(10×214)	(45)	(**4**)	(25×1000)	(4)	(**2**)
Australian	0.71s	**0.53**s	*Vehicle*	0.66s	**0.55**s
(15×690)	(25)	(**3**)	(19×846)	(16)	(**3**)
Splice	**0.39**s	0.93s	*Letter*	186.58s	**69.54**s
(61×1000)	(1)	(**1**)	(17×15000)	(35)	(**4**)

Table 4.2: *Performance of our FGMRES based Gaussian process regression against the CG based approach by Gibbs and Mackay (GM) in [34]; d is the dimension and N is the size of the dataset with a non-Gaussian kernel (Matern). Total time taken for prediction is shown here, with the number of iterations for convergence indicated within parenthesis. The mean error in prediction between the two approaches was less than* 10^{-6} *in all the cases.*

ica (the data map is shown in Fig. 3.7) used in Chapter 3. We look at the 7-day aggregate of the chlorophyll concentration, which is recorded on a grid of 416×600. However, this includes several locations with missing data or those located over land. This results in approximately $179,065 \pm 3,5405$ data samples per week.

It was observed that for each week's data the CG-based approach converges in 46 ± 12 iterations in $2,301 \pm 800$s, whereas FGMRES converges in just $\mathbf{3 \pm 1}$ iterations in $\mathbf{725 \pm 190}$s, thus resulting in over 3X speedup.

4.5 Conclusions and discussions

Krylov solvers are guaranteed convergence within N iterations, N being the number of samples. Therefore, when N goes high, reducing the number of iterations using a preconditioner at a marginal overhead improves computational performance. The improvement is significant for very large data sizes as is also observed in our experiments with *Sarcos* and kriging datasets. It can therefore be concluded that very large datasets mandate the use of efficient preconditioners to improve performance. When $\mathbf{K}$ is not regularized, the system is more prone for ill-conditioning, and the preconditioner becomes even more necessary and therefore will be required even for smaller datasets.

The key contributions of this work are as follows,

- A novel yet simple preconditioner is proposed to solve a linear system with a kernel matrix using flexible Krylov methods.
- A technique to accelerate the inner preconditioner system using truncated CG with fast matrix vector products was developed.
- Rules to select the preconditioner parameter were shown.

Although there has been a lot of research using Krylov approaches for kernel machines [34, 21, 60], to the best of our knowledge, this is the first paper to propose an accelerated preconditioner for Krylov methods for kernel systems. The proposed approach is not tied to any particular acceleration technique and would work with any generic MVP acceleration. The performance of the proposed approach is illus-

trated in various learning approaches with data sizes up to $200,000$, and there is an improvement of up to $\sim$ 10X in the number of iterations to converge and up to $\sim$ 3X in the total time taken compared to a conjugate gradient based approach, which complements the gains achieved via fast matrix vector products.

Chapter 5

Kernelized Rényi distance for subset selection

In the preceding chapters, we have looked at the GPU-based acceleration of kernel primitives and its extension to various classes of kernel regression problem. In this chapter, we explore a similarity measure based on the quadratic Rényi entropy. We utilize kernelization (Parzen window estimates) to evaluate the underlying distribution in a non-parametric fashion. The resulting computation cost is mitigated via GPUML.

Rényi entropy refers to a generalized class of entropies that have been used in several applications. We derive a non-parametric distance between distributions based on the quadratic Rényi entropy. The distributions are estimated via Parzen density estimates and the quadratic complexity of the resulting distance evaluation is mitigated with GPUML. This results in an efficiently evaluated non-parametric entropic distance - the kernelized Rényi distance or the KRD. We extend KRD to measure dissimilarities between distributions and illustrate its applications to statistical subset selection and dictionary learning for object recognition and pose estimation. In the next chapter, we will further extend the KRD into a similarity measure and show its application to speaker recognition.

5.1 Sample-based entropy estimation

The entropy of a distribution measures the amount of information contained by the distribution. The *Shannon entropy* is the most widely used entropic measure. For a random variable X whose probability distribution is $p(x)$, the Shannon entropy is given by,

$$H(X) = -\int p(x) \log p(x) dx \tag{5.1}$$

The Shannon Entropy is a specific case of a more generalized family of *Rényi entropies* characterized by a parameter α. The Rényi entropy of order α ($\alpha \geq 0$) is given by

$$H_\alpha(x) = \frac{1}{1-\alpha} \log \int p(x)^\alpha dx \tag{5.2}$$

As $\alpha \to 1$, the Rényi entropy reduces to the Shannon entropy (Eq. 5.1) in the limits as shown in [67]. The Shannon entropy of a joint probability distribution can be separated into the entropies of the individual random variables of the joint distribution. These properties, coupled with the analytical tractability of the Shannon measures for the commonly encountered parametric distributions, has made it the preferred choice for many problems. Despite this advantage, the Shannon entropy may be suboptimal in certain applications that require entropy estimation from samples [41].

Sample-based entropy estimation generally involves the pdf estimation ($p(x)$) followed by the entropy-integral approximation ($H(X)$ or $H_\alpha(x)$). The pdf estimation is much harder at higher dimensions, leading to an inconsistent entropy estimate which can be detrimental to the underlying application. However, it has been shown

that for a quadratic Rényi entropy ($\alpha = 2$), the pdf-estimation step can be bypassed by directly solving the integral with a kernel density estimate plug-in [67]. This results in a consistent estimator even for higher dimension, as is illustrated later in Section 5.2. Motivated by this, we consider the quadratic Rényi entropy and solve the integral with a kernel density estimate plug-in following [67]. We adapt the resulting distance measure to problems in speaker recognition, object recognition and pose estimation; improvements are seen in each case. Throughout this chapter (and the next one) the term *Rényi entropy* will refer to the quadratic Rényi entropy ($\alpha = 2$).

This chapter is organized as follows. The quadratic Rényi entropy is introduced, kernelized and accelerated in Section 5.2. The subset selection algorithm based on the kernelized Rényi distance is presented and validated in Section 5.3 before illustrating its applications in Section 5.4.

5.2 "Kernelization" of the Rényi Distance

The quadratic Rényi entropy (for $\alpha = 2$ in Eq. 5.2) is given by,

$$H_2(x) = -\log \int p(x)^2 dx. \tag{5.3}$$

If $p(x)$ is known, the entropy can be computed using the integral above. In many practical scenarios, the density is unknown and must be estimated from samples drawn from the distribution. There are parametric and non-parametric ways of estimating the density function. In the parametric case, a particular form for the density is assumed and the parameters associated with the form are estimated from

the samples, e.g. via the expectation-maximization algorithm. A non-parametric approach to density estimation uses a kernel window and estimates the density as a sum of kernel functions of the available samples from the distribution. Using kernel density estimation for $p(x)$ as in [82], we get

$$p(x) = \frac{1}{N}\sum_{i=1}^{N} K_h(x, x_i), \tag{5.4}$$

x_i indicates the sample location, $K_h(x, x_i)$ is a kernel function, quite often the Gaussian kernel,

$$K_h(x_1, x_2) = \frac{1}{h\sqrt{2\pi}} \exp\left(-\frac{|x_1 - x_2|^2}{h^2}\right), \tag{5.5}$$

with h the bandwidth that must be selected according to the data. This approach is preferred when the underlying distribution is unknown. Provided there are sufficient samples, a non-parametric approach provides unbiased estimates. Plugging-in Eq. (5.4) for $p(x)$ to Eq. (5.3), we get

$$\begin{aligned} H_2(x) &= -\log \int \left(\frac{1}{N}\sum_{i=1}^{N} K_h(x, x_i)\right)^2 dx \\ &= -\log \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \int K_h(x, x_i) K_h(x, x_j) dx. \end{aligned} \tag{5.6}$$

For the Gaussian kernel,

$$\int K_h(x, x_i) K_h(x, x_j) dx = \hat{K}_{\hat{h}}(x_i, x_j), \tag{5.7}$$

where $\hat{K}$ is also a Gaussian kernel with bandwidth equalling sum of the bandwidths of the two Gaussian kernels [118]. Using this relation in Eq. (5.6),

$$H_2(x) = -\log\left(\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} \hat{K}_{\hat{h}}(x_i, x_j)\right). \tag{5.8}$$

Consider two distinct distributions with densities p and q, with p defined by the set of data points, $D_p = \{x_{p1}, \ldots, x_{pN}\}$ and q defined by the set of data-points, $D_q = \{x_{q1}, \ldots, x_{qM}\}$, the distance between $p(x)$ and $q(x)$ is,

$$H_2(p\|q) = -\log\left(\frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\hat{K}_{\hat{h}}(x_{pi}, x_{qj})\right). \tag{5.9}$$

This is called Rényi cross-information potential [67]. This was first defined and analyzed by Principe et al. [67] and has since been used in several applications including clustering [43], visual tracking [120] and source separation [27]. We shall refer to this measure (Eq. 5.9) as the **Kernelized Rényi Distance (KRD)**.

The advantages of the KRD are:

(1) because the KRD uses a non-parametric on-the-fly density estimation, it does not require any parametric approximations for the distance evaluation;

(2) the KRD is symmetric unlike the popular KL-divergence measure;

(3) because it starts with Rényi entropy of $\alpha = 2$ it should exhibit faster convergence to the true measure for sampled data [40, 120].

The disadvantages of the KRD are its memory and computational complexity.

5.2.1 Accelerating KRD evaluation via GPUs

Evaluating the KRD between two distributions, each represented by N data-points, would require ($O(N^2)$) operations. It should be noted that the core computation in Eq. (5.9) is the summation of the Gaussian kernel. The key computation in KRD is the weighted summation of Gaussian kernel function, and we used GPUML [101] from Chapter 2 to accelerate KRD evaluations.

5.2.2 Inconsistency of sample-based KL divergence

Gockay et al. [35] observe that sample based estimation of the KL-divergence exhibits variability at higher dimensions because it is ratio-based (other ratio-based distances like Chernoff distance are also inconsistent at higher dimensions for sample based estimation). In this experiment, we illustrate this fact by using synthetic data and also show that the KRD measure (Eq. 5.9) does not exhibit such inconsistency.

In this experiment, we generated $10,000$ samples from two Gaussian distributions, $N(\mu, 0.25\mathbf{I})$ and $N(-\mu, 0.25\mathbf{I})$, where $\mu = \{1, \ldots, 1\}$ and $\mathbf{I}$ the identity matrix, for various data dimensions. KL divergence between two Gaussian distributions with means μ_1 and μ_2 and variances Σ_1 and Σ_2 is given by,

$$\begin{aligned} KL(p||q) &= \frac{1}{2}\ln\frac{|\Sigma_1|}{|\Sigma_2|} + \frac{1}{2}tr\left[\Sigma_1(\Sigma_1^{-1} - \Sigma_2^{-1})\right] \\ &+ \frac{1}{2}tr\left[\Sigma_2^{-1}(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T\right] \end{aligned} \tag{5.10}$$

This distance is made symmetric by taking the average of $KL(p||q)$ and $KL(q||p)$. Similarly the quadratic Rényi cross entropy between two Gaussian distribution is given by,

$$KRD(p||q) = \mathcal{N}(\mu_2|\mu_1, \Sigma_1 + \Sigma_2) \tag{5.11}$$

where $\mathcal{N}(x|\mu, \Sigma)$ is the evaluation of the Gaussian distribution with mean μ and variance Σ evaluated at x [7].

We evaluate the KRD between samples for all the dimensions along with the KL divergence based on the samples. For comparison we also evaluate the KL-divergence and quadratic Rényi entropic distance between the distributions based

on the first and second order statistics. As the dimension increases, the distance between distribution increases (as the means of the Gaussian are now more and more far placed) and is expected to be reflected in the corresponding measures. The normalized distance scores across dimension for various sample sizes (N) is shown in Fig. 5.1.

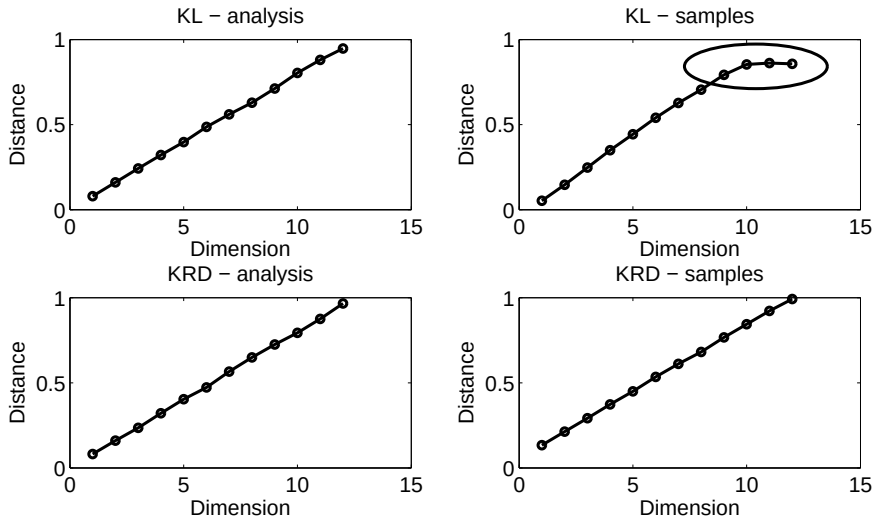


Figure 5.1: *Validation of the Kernelized Renyi Distance; Entropic distances between Gaussian distribution for various dimensions, distances evaluated analytically based on the underlying distribution and from samples (based on density estimates)*

It can be seen that the trend followed by the sample-based KRD score compares favorably with the statistics-based distance. However the trend of the sample-based KL divergence is skewed at higher dimensions illustrating the inconsistency. The variances of the corresponding KL sample-based distances are shown in Fig. 5.2, which indicates the associated instability. The variance of the other measures were $< 10^{-7}$ across several trials.

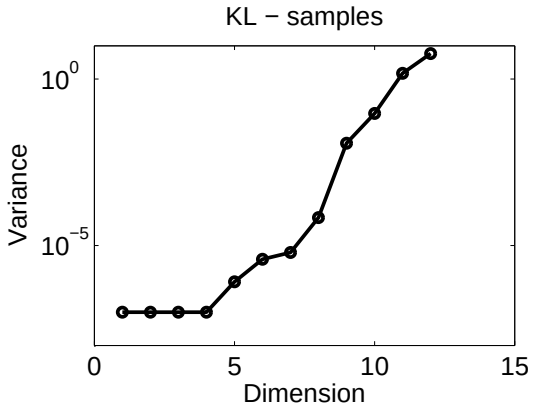


Figure 5.2: *Variance of the KL based on sample-based estimates*

It was observed that the sample based KL divergence estimates do exhibit the desired trend when estimated from a very large number of samples ($\sim 75,000$ samples for 15 dimensions). However, in a practical scenario, this critical sample size required to remove the underlying inconsistency in the trend is either unknown or is beyond the modeler's control.

5.2.3 Applications of KRD

There are numerous applications where the distance measure (equation 5.9) can be potentially applied. But however, we limit our experiments and discussion to the applications of the distance measure to statistical subset selection and similarity scoring. We shall discuss the subset selection in this chapter and defer the discussion on similarity scoring to the next chapter.

5.3 KRD for subset selection

The distance measure (Eq. 5.9) can be used in a greedy algorithm to extract a statistical subset such that the subset and the original data are as close as possible in the probabilistic space.

Existing algorithms for subset selection can be categorized into two types, greedy and clustering-based approaches. Greedy approaches [20, 50, 91] define a cost function to minimize and adds data to the subset that will minimize the cost. Clustering based approaches (eg. Vector Quantization) cluster datapoints in non-overlapping clusters and use the cluster centers as the low ranked representation. Both these approaches are well known for sparsification in learning and vision applications. Our objective is to use the KRD to develop a greedy algorithm to select a representative subset of a large dataset.

If the original distribution is denoted as $p(x)$, the subset selection can be formulated as forming a distribution $q(x)$ using data-points from $p(x)$ such that $p(x)$ and $q(x)$ are as close to each other as possible. In other words, we would want to add the next point in the subset to be drawn from the original set in such a way that $H_2(p\|q)$ is minimized by this addition. It is easy to see that for a direct use of the measure in Eq. (5.9) the subset will be clustered around the mode of the distribution. However for a subset to be actually representative of the data, it would be desirable to capture the significant outlier points as well. The distance measure in Eq. (5.9) is therefore modified as,

$$H_2(p\|q) = -\log\left(1 - \frac{1}{NM}\sum_{j=1}^{M}\sum_{i=1}^{N}\left(\frac{\hat{K}(x_{pi}, x_{qj})}{\hat{K}(x_{pi}, x_{pj})}\right)\right) \tag{5.12}$$

As mentioned before, the requirement on the subset selection is that the pdf defined from the subset should be as close as possible to the original distribution. Hence, in our KRD based subset selection, we minimize the distance between the subset distribution and the data distribution relative to the distance of the distribution with itself. This is done above by taking the ratio of the contribution of each training data element to the two distance measures. The subtraction from 1 is done to to formulate subset selection as a minimization. For numerical convenience, we clamp all ratios $\frac{\hat{K}(x_{pi},x_{qj})}{\hat{K}(x_{pi},x_{pi})}$ above 1 to 1 and set $\log 0 = 0$.

Greedy algorithms for subset selection fall into two categories; they either singly add data-points from the original set to a subset till the distance between the original and new distribution is less than a pre-defined threshold, or they add a pre-defined number of data-points incrementally. In this work, we will use the latter approach. Suppose a subset of size M needs to be extracted from a dataset of size N, the greedy algorithm would add the data points one-by-one. For each point, the distance measure is evaluated for all the points of the distribution from the distribution. Because of the use of fast Gauss summations, this step has a complexity of $\mathrm{O}(MN)$ and this is repeated for M points, thus leading to an overall complexity of $\mathrm{O}(M^2N)$. However, we have parallelized this computation efficient on a GPU using GPUML [101].

We exploit the facts that the distance measure in Eq. (5.12) is symmetric and that the influence of each sample point is additive (log is a monotone function). To minimize the distance at each iteration, we consider the contribution of each data-point in the original dataset to the distance, and add the point to the subset

Algorithm 4 *The greedy algorithm for subset selection using the distance measure*

Given: Data $D = \{x_1, \ldots, x_N\}$

Initialize subset I to be empty

for 1 to M (input subset size) **do**

Define set J = all elements in D not in I

Add an element (el) from J to I which minimizes

$H(p_D \| p_I)$ using Eq. (5.12)

Remove el from J

end for

Output I

that makes the largest relative distance contribution.

5.3.1 Validation: Kernel density comparison

In order to validate our approach to subset selection, we drew 2000 samples from the 15 normal density mixtures in [56]. We estimated the underlying density using the standard kernel density estimation, utilizing the entire set of drawn samples. We then used our KRD based subset selection to reduce the number of samples to 20% of the sample size, and estimated the kernel densities using this low ranked representation. The results for 6 of the 15 distributions are shown in Fig. 5.3. It can be seen that our low ranked estimates are similar to those obtained from the entire samples thus validating our approach further. Notice that the KDE on the entire dataset also misses some fine features because of the sample size.

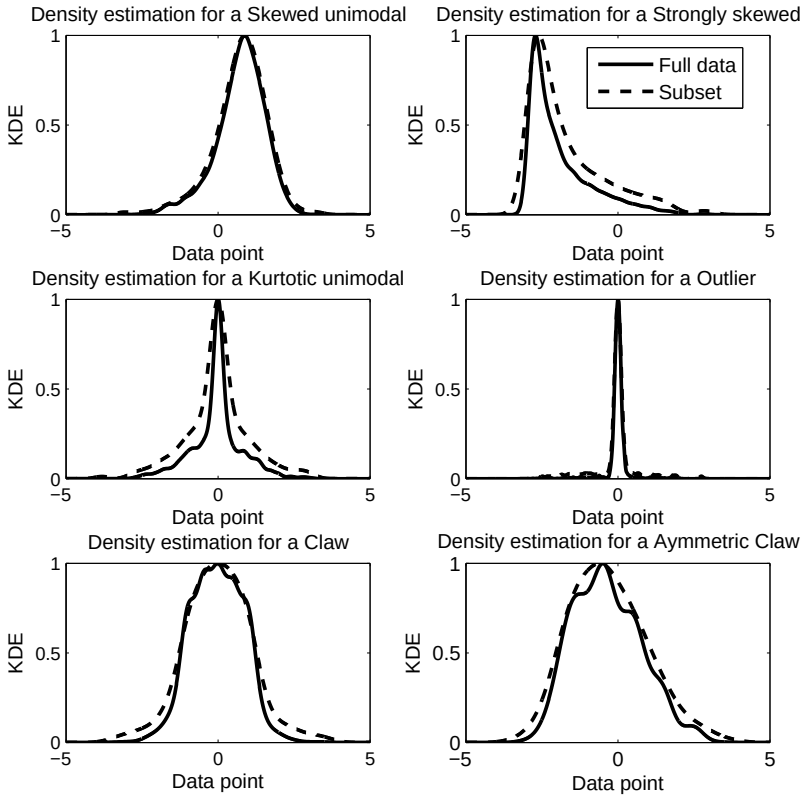


Figure 5.3: *Density estimates of the normal density mixtures in [56] using the entire samples and our low rank subset*

5.4 Applications of subset selection

The subset selection algorithm can be used in several applications.

With the improvements in learning algorithm, the complexity involved in learning has also increased along with the amount of data available. Hence sparse learning algorithms which use the sophisticated approaches with very few exemplar points are gaining popularity, for example, Support Vector Machine[8](SVM). Probabilistic algorithm like Relevance Vector Machine[109](RVM) and Gaussian Process

Regression [69](GPR) which not only provide the predictions, but also a confidence value for the prediction are gaining popularity. As seen in the Chapters 2 and 3, the Gaussian process regression is not sparse by nature. But several works have attempted to sparsify this; these approaches fall in three classes; 1.) a low rank approximation (chapter 8 in [69]); 2.) Learning from a subset of the original data like in [50, 20, 91]; 3.) learning using mixture of experts like approach. The proposed algorithm can be used in the second category here.

Vector quantization (VQ) is a well known approach in vision and audio applications. It has been used in object recognition [31] for learning a dictionary of codewords, which can later be used for forming histograms from objects. The histogram of the codewords are then used for training and classification of object categories. The key idea in the utilization of VQ in these applications is to find cluster centers which are then considered as representatives of the set. It is possible to use our subset selection approach in place of VQ.

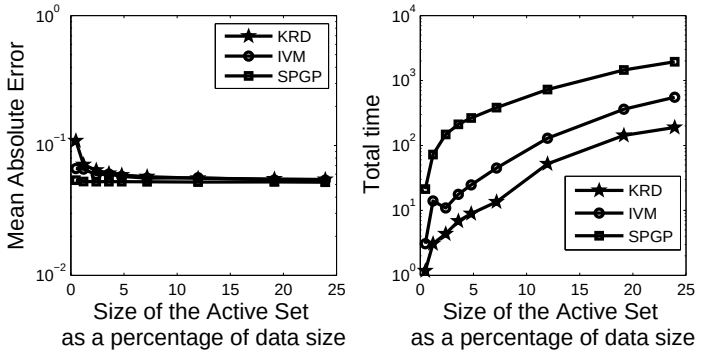
5.4.1 Gaussian Process Regression:

The core complexity in Gaussian process regression (Chapter 2) involves solving a linear system involving the kernel covariance matrix and hence is O(N^3). We have already seen the mitigation of this cost via GPU. An alternate approach to overcome this is to obtain a sparse representation (subset) of the original dataset which retains the information contained in the original data. For example, Online Gaussian Process (OGP) [20] uses a set of Basis Vectors (BVs) to train and predict

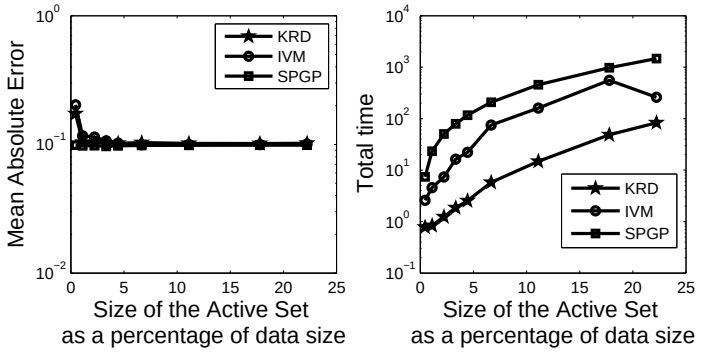
the GP model. Similarly, the Informative Vector Machine (IVM) [50] uses a KL-like distance measure to select a representative subset by approximating the posterior. Sparse Pseudo-input Gaussian processes (SPGP) [91] performs a sampling on the training points to obtain pseudo training data which is then used for training and prediction. Each of these approaches has a computational complexity of O(MN), where N is the size of the original data and M is the size of the subset. Along the same lines, we propose the use of our subset selection algorithm to obtain a subset of the training data, by using a combined input-output space, an idea inspired by [120] where a joint feature-spatial space is used for tracking. Once the subset was selected, we trained and predicted the Gaussian Process model [69].

In order to test the proposed algorithm with Gaussian process regression, we performed regression with two standard datasets, *Abalone* and *PumaDyn8NH* [110]. We compared the performance with popular sparse data selection methods for Gaussian processes - IVM and SPGP. Fig. 5.4 shows that our algorithm performed much faster than the other methods when applied to large datasets although the assymptotic complexity in our approach was O(N^2M) against (O(MN)) for IVM and SPGP.

It should be noted here that the error shown were absolute and not normalized. For example, the output variable in Abalone is its age and the errors obtained were of the order $1.5 - 2$. At these scales, our performance can be asserted to be comparable with other approaches. Further, the approaches with which we compared our method were tuned low-ranked approximations designed specifically for Gaussian process regression, thus our untuned subset selection performs on par with



(a) *Abalone*



(b) *PumaDyn8NH*

Figure 5.4: *Comparison of the performance of the training and prediction with our approach, Informative vector machine and Sparse Pseudo-input Gaussian Process with Abalone and PumaDyn8NH*

the other tuned approaches.

5.4.1.1 Pose Estimation:

Motivated by the superior performance of the KRD-based sparse GPR, we applied our approach to learn the head pose from human face images. Sparse regression based pose estimation has been done in several papers, for example, [55] uses RVM to train images to learn poses, [68] uses an online Gaussian process algo-

rithm to learn head pose from images. For this experiment, we used the PIE dataset [88] after annotating the image. For the purpose of this experiment, we considered only the horizontal orientations of the human face. The images were annotated with a score between -1 (left) to $+1$ (right) based on the horizontal orientation of the human face. A randomly selected class from the dataset is shown in Fig. 5.5 along with the score assigned to them.

Figure 5.5: *This is a randomly chosen class of pose images from the PIE dataset. The images were assigned scores of* {*-1,-0.75,-0.5,-0.25,0,0.25,0.5,0.75,1*} *from left-to-right*

Each image was projected onto a 30 dimensional subspace using PCA and were trained to learn the scores assigned to the image. Further, we also compared the results with popular sparse learning methods Relevance Vector Machine (RVM) (from [108] and Support Vector Machine (SVM) (from [16]. The error in prediction and performance are tabulated in table 5.1. In all the experiments, 90% of the images were used for training and the learning method was tested on the remaining 10%. 20% of the training data were selected by our method which was then used for training the GP model. Note that, both RVM and GPR are probabilistic regression approaches and provide a variance in prediction as well, a key difference from SVM. KRD-based GPR is faster than RVM. It is slower than SVM, but the additional

computational cost is to provide the variance in predictions.

Method	Mean Absolute Error in prediction	Time taken for prediction (seconds)
GPR	0.0261	20.7
RVM	0.0431	50.4
SVM	0.0755	9.9

Table 5.1: *Comparison of performance of our method with SVM and RVM for pose estimation. Each error entry gives the mean absolute error between the predicted face pose score and the actual score assigned to the image. Note that the prediction using RVM and GPR involved the evaluation of the variance (confidence) also, whereas the SVM computed only the predictions*

5.4.2 Visual words and object recognition

We applied our subset selection algorithm to object recognition. The bag-of-words approach [90, 31] have been widely used for object categorization because of its simplicity and good performance. The basic steps in bag-of-feature based object recognition can be summarized as:

1. Features are extracted from an image by either diving it into grids or using interest point detectors.

2. The features are then represented by a set of descriptors. One of the popular descriptors are the Scale-Invariant Feature Transform (SIFT) [53].

3. The next step is to generate a codebook from the descriptors. In this step, the feature descriptors are Vector Quantized (VQ) and the centers of the clusters are defined to be the codewords of the dictionary of object categories.

4. Features from the images can now be expressed as a histogram of all codewords in the dictionary.

5. The histogram is used to train a classifier for object categorization.

6. For an unlabeled image, the histogram of codewords is extracted, and then the trained classifier is used for classification.

We propose to replace the VQ step above with the KRD-based subset selection approach to get a representative set of the collection of descriptors. We show that by this approach, for comparable accuracy, there is a marked improvement in the time taken for dictionary formation. We used a standard k-means based vector quantizer for this experiment.

We use the SIFT descriptors of the image extracted after running an interest-point detector using the toolbox from [112]. In order to provide a basis for comparison, we also use a VQ based dictionary. Once the dictionaries are obtained, the histogram of codewords are extracted from the image. We use a 5-Nearest Neighbor classifier to compare the performance of the two dictionaries. The images used for the training and testing were obtained from the *Caltech-101* dataset [30].

In this experiment, we randomly choose classes from the dataset and extracted dictionaries using 5 images from each class with the two approaches mentioned. The dictionaries were used to obtain codeword histogram from each image. The trained histograms are then used to classify unseen test images using a 5 nearest neighbor search. We repeated the experiment for 2, 3, 4, 5, 6 and 10 class prediction, in each case the size of the dictionary was set at 30 times the number of classes trained. Table 5.2 shows the overall accuracy and time taken for dictionary formation for our approach and VQ based approach.

	VQ-based	Present method
2-class	77.8 (24.1s)	71.3 (18.7s)
3-class	62.3 (36.1s)	63.8 (26.7s)
4-class	78.4 (95s)	78.4 (83s)
5-class	61.4 (175.3s)	62.7 (103.6s)
6-class	63.4 (195.9s)	59.3 (114.2s)
10-class	47.8 (313.3)	52.7 (175s)

Table 5.2: *Accuracy of classification when objects from different number of classes were trained and predicted. The size of the dictionary was set to be* 30 *times the number of classes of object present. Each entry here indicates the over-all percentage of correct prediction, and the time taken for dictionary formation is given within braces*

It can be seen that, with comparable accuracy, our approach is much faster than the VQ based approach, especially as the number of classes increases. We have thus shown that the dictionary based on our method has comparable performance with the VQ based approach, but takes lesser time for dictionary formation.

5.5 Conclusion

In this chapter, we have developed a new information-theoretic distance measure based on KRD and used it to develop a subset selection algorithm. The subset selection algorithm was successfully applied to both synthetic and real problems (Gaussian process regression, and to replace vector quantization). Our approach, while being much more efficient, performed comparably or better than approaches previously used. In the next chapter, we will extend this measure to similarity scoring for a speaker recognition problem.

Chapter 6

Kernelized Rényi distance for similarity scoring

The KRD in Eq (5.9) represents the distance between two distributions $p(x)$ and $q(x)$. We looked at its application to subset selection in the last chapter. It is also possible to use the KRD as a similarity function, to measure similarities between classes (each represented by a set of feature points), by measuring the distance between feature distributions. We illustrate this idea in a speaker recognition problem in this chapter.

6.1 Speaker recognition

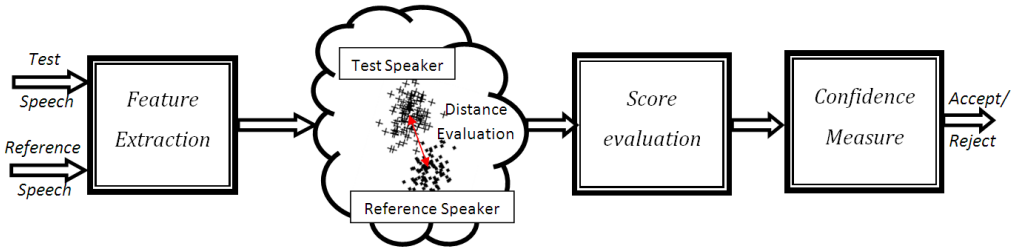


Figure 6.1: [color] *A modular representation of a generic speaker recognition system*

Fig. 6.1 shows a generic text-independent speaker recognition system that will be used in this chapter. Once a speech signal (both reference and test) is available, the first step in any recognition system is to extract features vectors from the signals. Once the features are extracted, there are many approaches to build

the speaker model. Gaussian Mixture Models (GMM) [73] build a semi-parametric model in the feature space, and are one of the widely used approaches in speaker recognition. Alternatively, it is possible to measure the distance between the feature vectors from the reference and test signals [6]. An advantage of such an approach is very low training time. We use such an approach in this work. As shown in Fig. 6.1, a scoring function is used to quantify the measured distance between the reference and test spaces. The main task of the scoring function in a recognition system is to find the similarity (or dissimilarity) between the reference and test signal space and quantify this using a matching score. The matching score can be used to authenticate based on a threshold, or to classify a speaker using k-nearest neighbor classifier.

There have been several information-theoretic and statistical measures that have been used to measure scores between speech signals. Second-order statistical measures [6] like sphericity and Gaussian likelihood have been used in speaker identification, which use only the mean and variance of feature vectors. Soong et al. [93] use a vector quantizer based codebook along with the Euclidean distance to compare speech signals. Information theoretic measures like KL-divergence and Bhattacharya distance have also been used in the speaker recognition framework [12]. However, the underlying feature distributions are assumed to be Gaussian in all these works. This can be limiting when the underlying distribution is seen to be very different from a Gaussian. Semi-parametric Gaussian mixture models [73] address this issue to some extent, and are popularly used in speaker recognition. A disadvantage with semiparametric and non-parametric approaches is the associ-

ated computational complexity, which make them undesirable for large problems. But however, in the proposed non-parametric distance (Eq. 5.9), we have already addressed the computational complexity using GPUs.

The KRD in Eq (5.9) represents the distance between two distributions $p(x)$ and $q(x)$. In order to use this as a scoring function in speaker recognition (Fig. 6.1), it is necessary to formulate the speech signals (reference and test) as samples from distributions. The feature selection in the recognition system extracts features from multiple overlapping frames of the speech signal. Suppose there are N and M overlapping frames in the reference and the test signal respectively, and d features are extracted, then we will have $N \times d$ vector representing the reference signal, and a $M \times d$ vector representing the test signal. We formulate this feature set to be samples drawn from the corresponding feature distribution of the speaker. Using Eq. (5.9), we can thus evaluate the matching score.

This chapter is organized as follows. The features and dataset used are described in section 6.2. The performance of KRD based similarity measure is illustrated for speaker identification in section 6.3 and for speaker verification in 6.4

6.2 Dataset and features

For this experiment, we used the speech signals from the TIMIT [22] database, which consists of data from 630 different speakers. Each sample for a speaker contains one sentence uttered by the speaker and there are totally 10 samples per speaker. We extracted 13 mel-frequency cepstral coefficients (MFCC) coefficients

from 25ms speech frames with 10ms overlap [26]. For all our experiments, we normalized the feature vectors to a set of zero-mean-unit-variance (except for the approaches that used only the first and second order statistics of the feature vectors). Although there are more complex sets of features used for speaker recognition (e.g., the above features can be augmented with velocity and energy and then pruned via split vector quantization), here we used just the MFCC because the objective was the comparison of distance measures. The method is of course generic enough to be extended with other features, and benefits obtained from GPU implementation will still be very significant.

6.3 k-NN for Speaker Identification

In speaker identification problem, the speaker is known *a priori* to be a member of a set of N speakers and a new test sample must be classified into one of N classes. In this experiment, we used the KRD measure with a 3-nearest neighbor classifier for speaker identification. We repeated the experiment with the GaussLL and VQ measures also using the 3-nearest neighbor classifier. We also built an SVM (with an RBF kernel) based speaker identification system [13] for comparison.

For each case, we use 5 samples for each speaker to do the training and test on the remaining samples. We evaluated the performance of each of the approaches for 10, 25, 50, 75, and 100-class scenarios. The classification results are shown in Table 6.1. It can be seen that the proposed approach performs better than the other approaches for all the cases.

Table 6.1: *Classification accuracy for various methods in speaker identification experiment.*

# of speakers	VQ	GaussLL	SVM	KRD
10	**96.00**	94.00	94.00	**96.00**
25	90.40	91.20	82.40	**92.00**
50	70.67	73.87	66.80	**78.40**
75	64.40	71.60	61.07	**74.40**
100	54.80	63.20	55.80	**64.80**

6.4 Likelihood Ratio for Speaker Verification

Speaker verification system accepts a sample X as a speaker S if the likelihood ratio $\frac{P(X|S)}{P(X|S')} > T$, where T denotes a threshold. The likelihood $P(X|S)$ denotes the probability that the features from the sample X were generated by speaker S. Similarly, $P(X|S')$ denote the probability the features are from an imposter. The threshold T can be adjusted so that the false acceptance rate (FAR) (an imposter being identified as a speaker) and the false rejection rate (FRR) (a speaker being rejected as an imposter) are equal. We used this Equal Error Rate (EER) criterion to evaluate the performance of our measure.

We compared our scoring function with the Gaussian-likelihood measure [6] (GaussLL), Euclidean distance between vector quantized codebook [93] (VQ), KL-based measure [12] (KL_a), KL-scores evaluated from the samples (KL_s), and GMM-UBM based score [73]. The Matlab *kmeans* function was used to build the codebook

of size 50. The GMM was built using the statistical toolbox in Matlab, and number of mixtures was chosen to be 32 with diagonal covariance for each speaker. The universal background model [73] (UBM) for the imposter was built by collecting feature samples from a large number of speakers in the database. For the GMM, the UBM had 256 mixtures, whereas for other measures the entire set of UBM samples were used.

We evaluate each of the above scores for a test signal with respect to the reference speaker and imposter speaker models and compute the ratio between the two, which is then used for threshold comparisons. The equal error rate obtained in this way is reported in Table 6.2 for each of the scores. It can be seen that the proposed scoring function outperforms the other approaches in all the cases.

In Table 6.2, we have also reported the average time taken to evaluate the score between two sets of feature vectors (speaker/imposter). The measures *GaussLL* and KL_a take the least time. However, these measures use only the first and second order statistics for score evaluation and hence inexpensive to compute. While our score is more complex, it still takes lesser time than all advanced approaches (VQ, GMM, and KL_s).

6.5 Conclusions

We have adapted the KRD into a scoring function and used in a speaker verification and identification system. The results compare favorably with other

Table 6.2: *EER for various methods in speaker verification experiment. Time reported is the average time of one score evaluation. Time to build the imposter models for GMM and VQ is not included.*

	VQ	KL_s	KL_a	GaussLL	GMM	KRD
Time	0.7s	4.5s	0.03s	0.04s	0.4s	0.16s
EER	5.33	6.67	6.67	6.00	8.00	**4.67**

similarity score based approaches. However, *state-of-the-art speaker recognition system yields an equal error rate of* 1.75 *with the TIMIT database taking around* 1.2s *per speaker*. The state-of-the-art system is based on a Gaussian Mixture Modeling (GMM) with a Universal Background Model (UBM) and speaker adaptation [73] and generalizes well to limited training data. This is key aspect lacking in the KRD-based classifiers and also exists for other similar classifiers. In the next chapter, we explore the GMM-UBM system briefly and use it to propose a new learning solution in this space that generalizes well for limited training data and compares well to the state-of-the-art systems.

Chapter 7

A partial least squares framework for speaker recognition

In the last chapter, we formulated the problem of speaker recognition as a task of measuring the distance of a test distribution from a target and imposter distribution to make a decision. While this was promising, it did not compare well with the state-of-the-art recognition system. In this chapter, we explore the state-of-the-art recognition system to extend it to a discriminative learning framework.

7.1 Speaker adaptation and supervectors

State-of-the-art speaker recognition systems use a Gaussian mixture model (GMM) to represent each speaker. Given, unknown speaker features x, the likelihood for a given speaker s_i is given by,

$$p(x|s_i) = \sum_{i=1}^{K} \pi_k \mathcal{N}(x|\mu_{ik}, \Sigma_{ik}), \tag{7.1}$$

where $\{\mu, \Sigma, \pi\}$ are the GMM parameters and $\mathcal{N}(x|\mu, \Sigma)$ indicates a Gaussian distribution with mean μ and variance Σ evaluated at point x. To account for limited training data available, the problem is cast into a framework in which differences from a universal background model (UBM) are used to adapt speaker-specific GMMs [73]. The universal background model is built by pooling features from a large number of speakers and denotes an average speaker. Training samples from a particular speaker is used to adapt this model toward the speaker in a MAP sense [73]. This

is illustrated in Fig. 7.1, where the UBM is colored in blue and the adapted speaker in red.

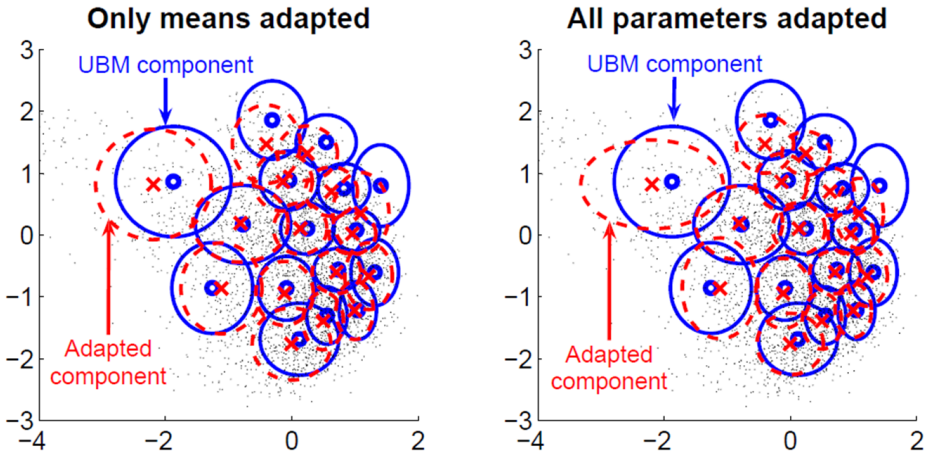


Figure 7.1: [color] *GMM Speaker Adaptation with Universal Background Model*

More recently, the problem has been transformed into a task of learning the between-class separability in a supervector setting [48]. The supervectors are obtained by concatenating the centers of the adapted speaker GMMs into a single vector (Fig. 7.2) and this results in a single fixed-lenght feature vector for a speaker utterance of any length. The objective in the supervector space is to discriminate between a speaker and imposters by accounting for the speaker variability while ignoring nuisance information. Commonly, only a few (often one) speech samples from a very large speech database belong to the target speaker, which necessitates use of the method capable of learning from a few samples in a very high dimensional space. Different approaches such as GMM likelihood ratios [73] and support vector machines [15] have been explored previously.

Several learning techniques have been used to tackle similar scenarios in other

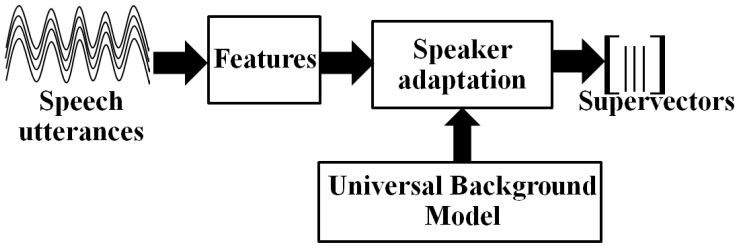


Figure 7.2: *GMM to supervectors*

domains. One approach that was originally developed in chemometrics is partial least squares [75] and its kernelized version [77]. *Partial least squares (PLS)* [75] techniques are a wide class of methods for modeling relations between sets of observed variables by means of latent variables. PLS has been used for regression, classification, and dimensionality reduction. The underlying assumption in PLS is that the observed data is generated by a system/process that is driven by a small number of latent variables.

PLS is often used as a dimensionality reduction technique and therefore draws comparisons with principal component analysis (PCA) and linear discriminant analysis (LDA). PCA is an unsupervised dimensionality reduction algorithm, which results in a single projection irrespective of the task. LDA is a supervised dimensionality reduction technique that results in different subspaces for different tasks. PLS is similar to LDA in this sense. But unlike LDA, PLS is not limited by a projection space dimension of $c-1$ (where c is the number of classes). A detailed comparison of PLS, PCA, and LDA is presented in [77]. PLS based techniques have been very successful in the fields of chemometrics and bioinformatics. Recently, PLS has been adapted to image processing and computer vision problems (such as

human detection and face recognition) [81] and was shown to greatly improve the performance, especially for 2-class problems.

Motivated by this, we explore here a partial least squares based framework for speaker modeling and recognition in the supervector space. Extension to handling nuisance parameters is discussed in the next chapter. This chapter is organized as follows. In Section 7.2, we introduce the PLS framework and its adaptation to speaker recognition. The framework is accelerated on a GPU in Section 7.3. We describe our experiments and discuss results in Section 7.4.

7.2 Partial Least Squares

Denote a d-dimensional supervector by x and the corresponding speaker label by y. Essentially, x is the feature (super)vector (input variable) and y is the speaker identity (output variable that has to be learned). Assume that the total number of speakers is N and denote the $N \times d$ matrix of supervectors by X and the $N \times 1$ vector of labels (1 for speaker and -1 for imposter) by Y. Given the variable pairs $\{x_i, y_i\}, i = 1, \ldots, N$ ($x \in R^d, y \in R$), PLS aims at modeling the relationship between x and y using projections into latent spaces. While a detailed analysis of PLS can be found in [75], we provide a brief overview here. PLS decomposes X and Y as

$$X = TP^T + E, \tag{7.2}$$

$$Y = UQ^T + F, \tag{7.3}$$

where T and U ($N \times p$, $p < d$) are the latent vectors, P ($d \times p$) and Q ($1 \times p$) are the loading vectors, and E ($N \times d$) and F ($N \times 1$) are residual matrices. PLS is usually solved via the *nonlinear iterative partial least squares (NIPALS) algorithm* [75] that constructs a set of weight vectors $W = \{w_1, w_2, \ldots, w_p\}$ such that

$$\max[\text{cov}(t_i, u_i)]^2 = \max_{|w_i|=1}[\text{cov}(Xw_i, Y)]^2, \tag{7.4}$$

where t_i and u_i are the i^{th} columns of T and U respectively and $\text{cov}(t_i, u_i)$ indicates the sample covariance between latent vectors t_i and u_i. Maximizing the covariance in the latent vector space is equivalent to maximizing discrimination in the same space; in other words, for a particular speaker, PLS learns a subspace in which the speaker latent vectors t_S are well separated from the imposter latent vector t_I. This is illustrated in Fig. 7.3. Thus, PLS learns a unique latent space for each speaker. After extraction of latent vectors t_i and u_i, the matrices X and Y are deflated by subtracting their rank-1 approximation based on t_i and u_i:

$$X \leftarrow X - t_i p_i^T; \qquad Y \leftarrow Y - u_i q_i^T. \tag{7.5}$$

This step removes any information captured by t_i and u_i from X and Y. The process is repeated till a sufficient number of latent vectors is obtained. This number is determined via standard cross-validation techniques [81].

It can be shown [75] that the weight w in NIPALS corresponds to the first eigenvector of the following eigenvalue problem and the NIPALS is just a mirror of the popular power iterations for finding the dominant eigenvectors,

$$[X^T y y^T X]w = \lambda w. \tag{7.6}$$

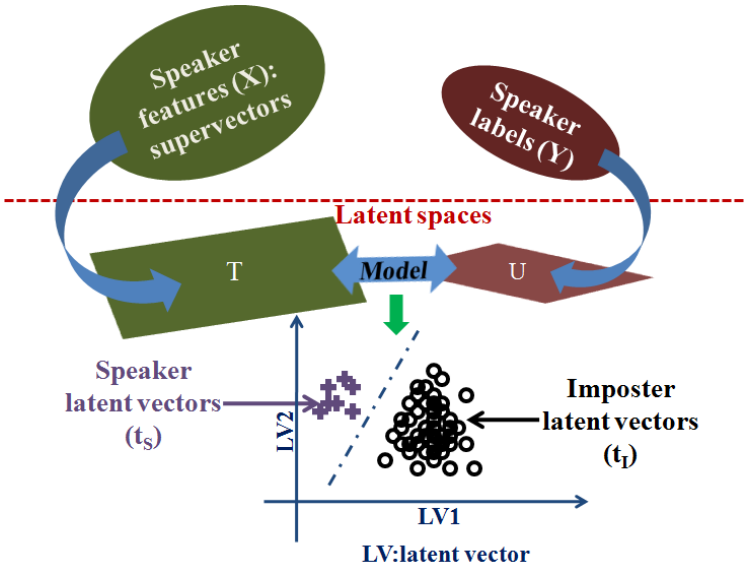


Figure 7.3: [color] *Partial Least Squares (PLS) latent spaces for speaker recognition.*

Because the rank of the above system is limited by the number of samples N, $N < d$ yields a few dominant eigenvectors and hence PLS works best in this scenario.

The weight matrix W can be used for dimensionality reduction, and the resulting projection can be used with any standard classifier to model a target speaker. However, it was observed that the performance was not as good as the alternative presented below. We instead use PLS in a regression framework that implicitly utilizes the PLS weights W obtained from the NIPALS algorithm.

7.2.1 PLS Regression

Substituting the w from Eq. (7.4) in Eq. (7.2), we get

$$XW = TP^TW + E \Rightarrow T = XW(P^TW)^{-1}. \tag{7.7}$$

Now, U can be written in terms of T [75] as $U = TD + H$, where D is a diagonal matrix and H is the residue. Eq. (7.3) now becomes

$$Y = TDQ^T + HQ^T + F = XW(P^TW)^{-1}DQ^T + \bar{F}, \tag{7.8}$$

and we get the PLS regression:

$$Y = XB + G; \qquad B = W(P^TW)^{-1}DQ^T, \tag{7.9}$$

where B is the set of PLS regression coefficients. This regression framework directly provides the way to compute the matching score for seamless speaker discrimination, eliminating the need for a separate classifier. It also utilizes the latent structure learnt by NIPALS algorithm better – the regression coefficients weight the supervector centers that discriminate the current speaker against imposters more than other centers. Hence, the regression coefficients are unique to each speaker. Note that, although PLS is used widely a dimensionality reduction technique, we use a PLS-based regression technique, and the dimensionality reduction is not used explicitly for speaker modeling.

In our work here, we first train the GMM UBM using a large amount of data. Then, we create a specific GMM for each speaker in the database by adapting the UBM using the speaker (training) utterances. Then, the speaker supervector [48] is created by concatenating the means of the speaker GMM. Note that the whole training utterance is represented by one point in the supervector space. We then learn the PLS regression model using a one-vs-all scheme. Finally, we perform the scoring and Z-normalize the output scores [5] using a large number of non-target speakers (imposters). These steps are summarized in Fig. 7.4.

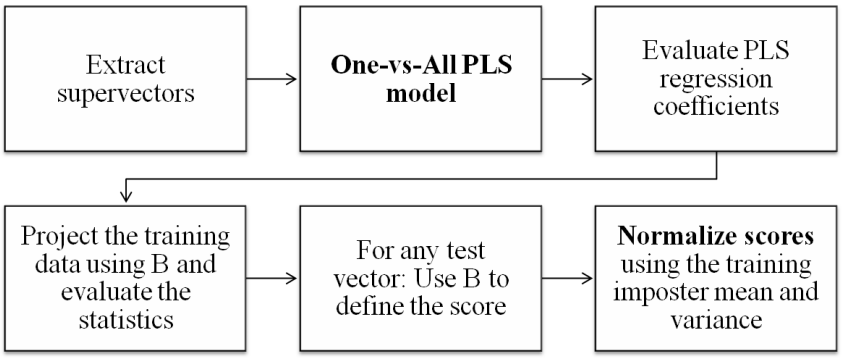


Figure 7.4: *Schematic of the proposed Partial Least Squares (PLS) technique for speaker recognition.*

The beneficial properties of the proposed PLS framework for speaker recognition can be summarized as follows:

1. It is a discriminative technique (like SVM); hence, the performance should improve as the amount of speaker training data increases.

2. SVM learns a separating hyperplane between speaker and imposter supervectors, whereas PLS learns discriminative projection that maximizes the covariance of supervectors and speaker labels in the projected space. PLS regression weights the supervectors based on these projections to score each utterance.

3. The computational cost of PLS is O(Nd) against O(N^2d) for SVM, where d is the supervector dimension and N is the number of supervectors.

4. The PLS technique used here is linear. Non-linear PLS can potentially be done by using the kernel-trick [77]. However, this direction will be explored in

the next chapter.

7.3 Accelerating PLS

Despite the success of PLS, its O(Nd) computational cost may not work well for very large sample sizes or feature dimension. We address this scalability issue via use of graphical processors. The NIPALS algorithm is shown in Alg. 5 and comprises two main steps; in the first step, the weight w is evaluated according to Eq. 7.4. Once w is obtained, NIPALS performs a deflation of the X and Y matrices, which is a rank-1 update such that any information captured by w is removed from X and Y. If the desired latent space dimension is not achieved, the algorithm returns to the first step to evaluate a new w.

The NIPALS algorithm (Alg. 5) involves a number of linear algebra operations on the feature matrix X and response variable Y. The asymptotic space and time complexity is O(dN) (assuming $f \leq d$). It is not possible to do away with O(dN) space requirement because this is required to store the feature matrix. However, the time complexity can be addressed with efficient parallelization strategy using graphical processors.

CUBLAS is an implementation of BLAS (Basic Linear Algebra Subprograms) on top of the NVIDIA CUDA driver. The library is self-contained at the API level, that is, no direct interaction with the CUDA driver is necessary. CUBLAS attaches to a single GPU and does not auto-parallelize across multiple GPUs. The basic model by which applications use the CUBLAS library is to create matrix

Algorithm 5 *Nonlinear Iterative PArtial Least Squares (NIPALS) algorithm (with CUBLAS)*

Given: $N \times d$ feature samples X and $N \times f$ response variable Y

repeat

Given: $N \times d$ Feature samples X and response variable Y

Allocate GPU memory for X and y and transfer data to GPU [`cublasAlloc,cublasSetVector`]

repeat

$w = X^T u/(u^T u)$; $\|w\| \rightarrow 1$: [`cublasSgemv,cublasSscal,cublasSnrm2`]

$t = Xw$: [`cublasSgemv`]

$c = Y^T t/(t^T t)$; $\|c\| \rightarrow 1$: [`cublasSgemv,cublasSscal,cublasSnrm2`]

$u = Yc$: [`cublasSgemv`]

until Convergence

$p = X^T t$: [`cublasSgemv`]

Deflate $X : X \leftarrow X - tp^T$ and $Y : Y \leftarrow Y - tc^T$: [`cublasSgemv`]

until Required number of factors are obtained

and vector objects in GPU memory space, fill them with data, call a sequence of CUBLAS functions, and, finally, upload the results from GPU memory space back to the host. To accomplish this, CUBLAS provides helper functions for creating and destroying objects in GPU space, and for writing data to and retrieving data from these objects. CUBLAS offers best speedup for BLAS2 (matrix-vector operations) and BLAS3 (matrix-matrix operations) operations.

NIPALS has several BLAS1, BLAS2 and BLAS3 tasks. Therefore, the best

computational performance would result if BLAS2 and BLAS3 are performed on GPU and BLAS1 on CPU. But, this would result in several data movements back and forth between CPU and GPU, and can cost heavily in access times. Therefore, in our GPU-based NIPALS we perform all blas operations on the GPU. Such a strategy would be advantageous because the BLAS2 and BLAS3 speedups are significant and the savings on the memory transfer times is big enough to weigh over BLAS1 disadvantages. The pointers to various CUBLAS function that can be used at various steps in NIPALS is enlisted in Alg. 5.

We illustrate the GPU acceleration of NIPALS on the INRIA dataset used in [81]. Fig. 7.5 shows the speedup of our GPU-based NIPALS for various sample and feature sizes. Although there is considerable speedup for lower datasize/dimension, significant speedup is obtained for large datasize/dimension indicating its utility for large datasets.

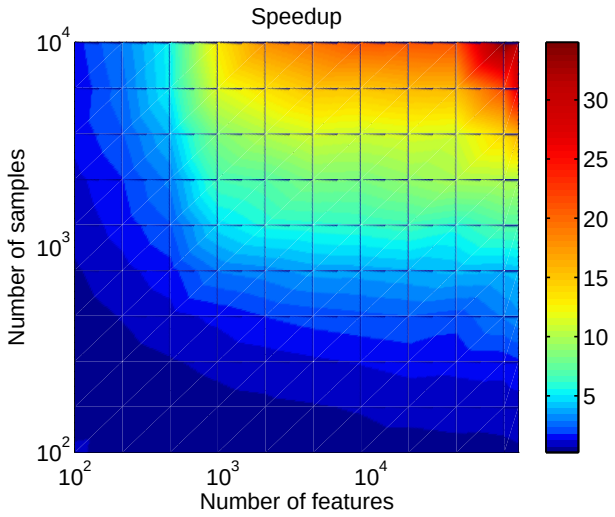


Figure 7.5: [color] *Speedup obtained by the GPU-based NIPALS for various sample sizes of different feature dimensions*

7.4 Experiments

We performed experimental evaluation of the proposed method on the *core (short2-short3) test set* and *8conv-short3 test set* in the NIST SRE 2008[1] evaluation dataset. The dataset is grouped into 8 trial conditions: C1: interview speech (IS) both for training and testing (BTT); C2: IS, using the same microphone for training and testing; C3: IS, using different microphones for training and testing; C4: IS for training, telephone speech (TS) for testing; C5: TS for training, noninterview microphone speech for testing; C6: TS BTT; C7: English TS BTT; and C8: English TS BTT by native English speakers. For all experiments, we used 19 MFCC features along with their deltas.

We compared performance of the PLS-based approach against the GMM/UBM based system [73] and GMM-supervector-kernel based SVM [15]. The libSVM package was used for our SVM runs. The GMM/UBM code was developed in house and validated against results reported in NIST SRE 2006. Note that since nuisance attributes are not being modeled, the GMM/UBM EER is relatively high compared to SRE 2008 results (where nuisance corrections based on JFA were applied).

7.4.1 Supervector dimensions

It was observed that 4096-center GMM gave the best performance with core set, while 2048-center model was best for 8conv-short3 set. With SVM, these numbers are 1024 and 512, respectively; and with PLS, 512 and 256, respectively. While

[1] `www.itl.nist.gov/iad/mig/tests/sre/2008/`

a larger number of GMM centers leads to severe over-fitting to the background data (which helps GMM capture background characteristics better but does not provide room for supervector based discrimination), very few centers lead to severe under-fitting and the resulting GMM models do not generalize well to test conditions. This is the reason PLS works best with moderate number of GMM centers, which also reduces the computational load by an order of magnitude.

7.4.2 Single training utterance

In the core set, there is only one training utterance per speaker. There are 1270 male and 1993 female speakers (3263 total) and 98776 trials. Each trial belongs to one or more of 8 conditions outlined above. The DET curves for all 8 conditions are shown in Fig. 7.6 and the equal error rates are listed in Table 7.1.

Note that having only one training utterance per speaker does not provide enough data for discriminative approaches like SVM and PLS; the GMM/UBM system is likely to perform better in this case. In spite of that, the PLS framework outperforms GMM/UBM in conditions $2, 5-8$ (5 out of 8) and is comparable for condition 4.

As mentioned, the PLS framework makes use of intra-speaker variability. Therefore, the performance is expected to improve if more supervectors belonging to the target speaker are available. Ideally, speaker utterances should be recorded across various nuisance conditions, which will enable PLS to truly capture speaker-related information and reject channel-related one. Alternatively, we explored simple

	GMM	SVM	PLS	PLS 2-splits	PLS 4-splits
C1	**13.84**	19.73	18.43	18.02	17.13
C2	6.15	12.73	**3.38**	**3.38**	3.64
C3	**13.54**	19.49	18.39	18.08	17.13
C4	**21.31**	23.63	22.48	22.79	22.79
C5	19.03	24.34	13.90	**13.64**	13.90
C6	13.48	18.44	10.13	**9.60**	9.77
C7	10.69	15.41	6.52	**5.51**	5.92
C8	10.42	16.34	6.61	**5.47**	5.98

Table 7.1: *Equal-error-rates obtained with PLS (with/without data splitting), SVM, and GMM across various condition for the NIST 2008 core set. Note: there is no nuisance attribute compensation.*

mechanism of splitting the training data to create multiple supervectors per utterance. Note that this does not guarantee the availability of training vectors across nuisance conditions. However, it was observed that the PLS performance indeed improved significantly with 2-way split of the training data, although there is no further improvement with 4-way split. The DET curves for these cases are also shown in Fig. 7.6.

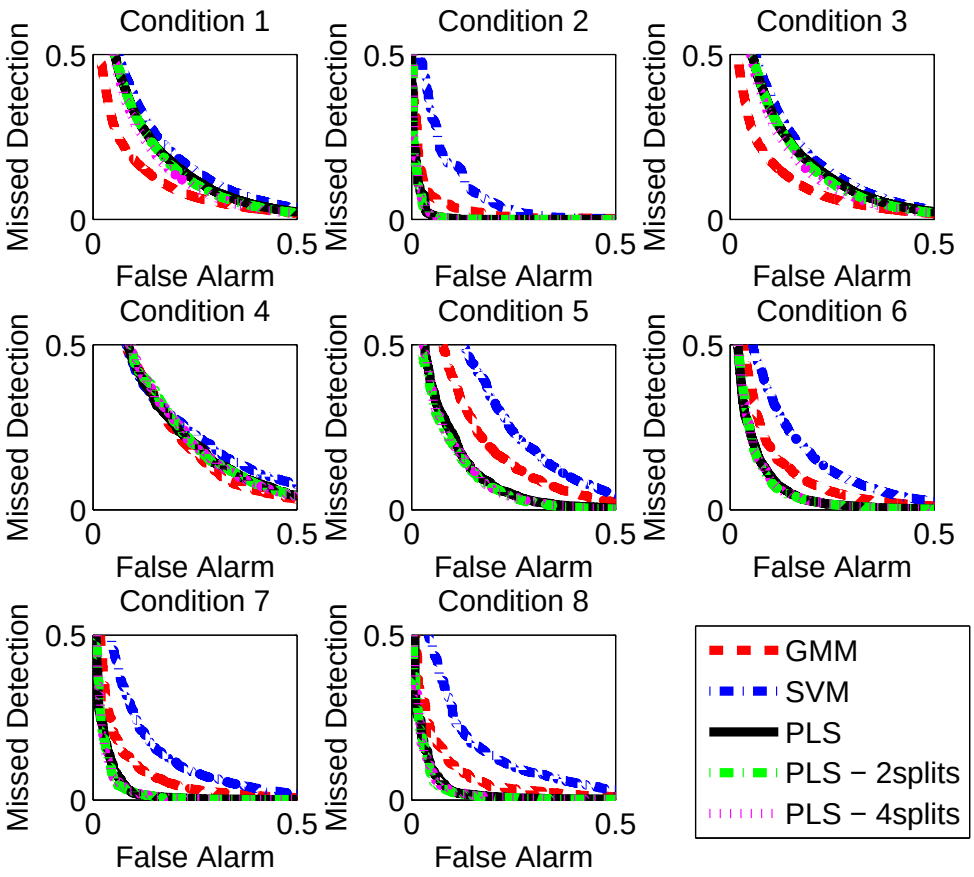


Figure 7.6: [color] *Performance of PLS against SVM and GMM baseline systems on the NIST 2008 core set.*

7.4.3 Multiple training utterances

The 8conv-short3 set consists of 8 training utterances per speaker. There are 240 male and 395 female speakers (635 total) and 16570 trials. There are no trials corresponding to conditions C1 through C4, as all training data is telephone speech.

We compared the performance of PLS-based speaker recognition against GMM/UBM and SVM baseline systems, and the DET curves are shown in Fig. 7.7 with the corresponding equal error rates in Table 7.2. It can be seen that PLS outperforms other

systems in all conditions.

	GMM	SVM	PLS
C5	10.98	10.52	**8.63**
C6	6.66	5.62	**4.30**
C7	4.82	3.94	**2.41**
C8	5.27	3.76	**3.02**

Table 7.2: *Equal-error-rates obtained with PLS, SVM and GMM across various condition for the 8conv-short3 set. Note: there is no nuisance attribute compensation.*

7.4.4 Effect of training sample size per speaker

Because the 8conv-short3 set contains 8 training utterances (and therefore 8 supervectors) per speaker, it also provides a good framework for evaluation of the recognizer performance dependence on the amount of training data. We have trained each of our recognition systems with $1, 2, \ldots 8$ utterances per speaker; the corresponding results are shown in Fig. 7.8(a). It can be seen that all 3 system show improved performance with the increase in the number of training speaker supervectors. However, unlike GMM and SVM, PLS performance does not saturate but instead continues to decrease. This is because PLS relies on the intra-class variance to determine the projection; therefore, having more training data implies better intra-speaker variance estimate and better performance.

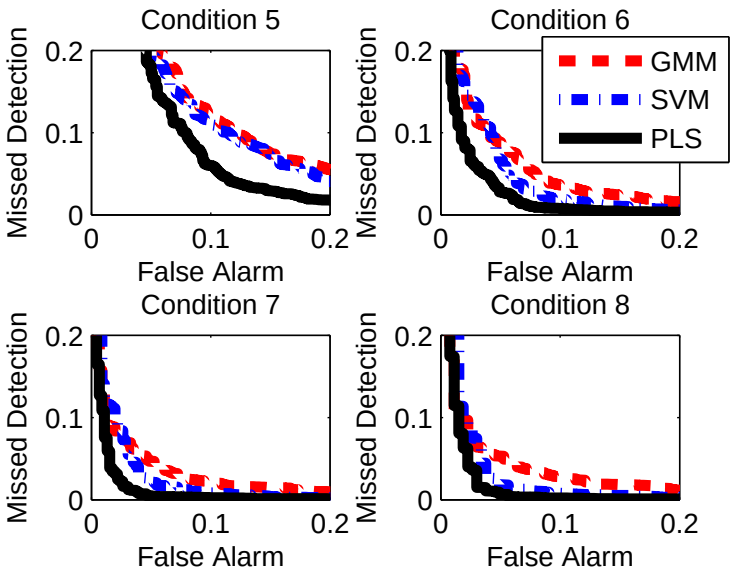


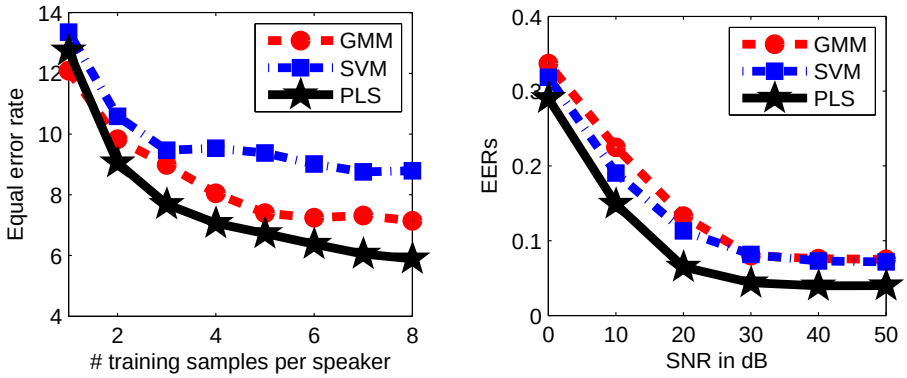
Figure 7.7: [color] *Performance of PLS against SVM and GMM baselines on 8conv-short3 set.*

7.4.5 Noise robustness of PLS

To evaluate PLS robustness to noise, we added Gaussian noise to test samples in the 8conv-short3 set (male only) and evaluated the performance of all three recognition systems. The results are shown in Fig. 7.8(b). It can be seen that additive noise decreases the performance for all systems, but PLS still outperforms both SVM and GMM.

7.5 Conclusion

We have applied a PLS latent vector framework to the GMM supervectors in speaker recognition and have shown that it outperforms the baseline GMM/UBM and SVM systems on NIST 2008 SRE dataset in most conditions. However, the



(a) *Number of training points* (b) *Noise robustness*

Figure 7.8: [color] *EER for PLS, SVM, and GMM/UBM systems under various scenarios.*

PLS-speaker scoring is asymmetric, viz. given 2 samples A and B, training a PLS model on A and testing on B does not yield the same decision as training a model on B and testing on A, which is desirable in many recognition engines. We shall address this in the next chapter.

The PLS system we currently have does not account for nuisance parameters (channel and session variability); therefore, our baseline systems also did not include nuisance parameter elimination for fair comparison. Nuisance compensations with PLS will also be discussed in the next chapter.

Chapter 8

Kernel PLS framework for speaker recognition

In the last chapter, we introduced a PLS framework for speaker recognition sans nuisance compensation. Within the PLS framework, it is possible to compensate nuisance directions either at the PLS end, or at the supervector end. We had explored both options, however the latter was more successful; we describe this in this chapter.

Recently, several approaches have been tested to make the GMM-based speaker recognition robust to session and channel variabilities, including JFA technique [10] [47] and the i-vectors framework [24]. JFA learns two subspaces of maximal channel-related and speaker-related variabilities and attempts to jointly learn the directions along these subspaces for any given supervector. On the other hand, i-vectors encapsulate the directions in a total variability space. The i-vectors are smaller in dimension compared to the GMM-supervectors and thus provide an abridged representation of the utterance.

Given the i-vector representation, the key problem is to develop learning techniques that distinguish target and non-target trials in the i-vector space. Generative PLDA models [45], discriminative SVMs [14], and CDS classifiers [84] have been studied for speaker recognition using i-vectors. We have earlier extended a linear PLS framework in supervector-space [104] to a kernel framework to best model the

non-linear i-vector space[100].

Discriminative frameworks in speaker recognition fall under two categories. The first class of approaches models each speaker independently against a set of background speakers [14]. While such a independent modeling is preferred in many practical recognition systems, this often suffers from an imbalance in the number of positive and negative examples available for training. Also, the scoring is generally asymmetric (while such a problem may exist for certain generative models as well, popular methods like JFA, PLDA, etc have symmetric scoring). Alternate methods attempt to learn a pair-wise similarities between a train utterance and a test utterance e.g. [84, 9]. While such an approach does not allow for explicit independent speaker modeling, it does not suffer from data imbalance and asymmetric scoring due to pair-wise considerations. In this chapter, we propose to combine both these class of approaches in to a single hybrid framework by developing KPLS-based discriminative based "one-shot similarity" framework.

One-shot similarity framework [117, 116] has been proposed in the context of face recognition. Given two (feature) vectors, the one-shot-similarity reflects the likelihood that each vector belongs the same class as the other vector and not in the class defined by a set of negative examples. The potential of this approach has been explored widely in computer vision [116, 117].

This chapter is organized as follows. In Section 8.1, the factor analysis framework is introduced and extend to obtain total variability framework with i-vectors. KPLS framework is introduced and adapted to the speaker recognition problem in Section 8.2. The one-shot similarity scoring is detailed in Section 8.3. Section 8.4

discusses the recognition results on NIST SRE 2010 data and includes comparisons against several state-of-the-art systems.

8.1 Joint Factor Analysis and the i-vectors

While the MAP adaptation of the UBM works well for speaker recognition, it fails in the presence smaller training u While the MAP adaptation and supervectors work well for speaker recognition, it fails in the presence smaller training utterances where the data sparsity prevents some components of the UBM from being adapted. Further it does not address any nuisance compensations in the supervector space. Joint Factor Analysis (JFA) [46] attempts to address this by correlating the various Gaussian components of the UBM. The key assumption in JFA is that the intrinsic dimension of the adapted supervector is much smaller than $N \times d$. JFA breaks down the speaker- and channel- dependent supervector M into two components,

$$M = s + c, \tag{8.1}$$

where s is the speaker dependent part, and c is the channel dependent part given by,

$$\begin{aligned} s &= m + Vy + Dz \\ c &= Ux \\ M &= m + Ux + Vy + Dz \end{aligned} \tag{8.2}$$

where $\{m, U, V, D\}$ are the hyper-parameters of the JFA model, which are estimated via Expectation Maximization (EM). The rationale behind this equation is that

aside from the offset m (UBM), the mean supervector is the superposition of three fundamental components with rather distinctive meaning. The component that lives in U is used to denote undesired variabilities in the observed vectors (e.g. channel related variability). The component living in V is used to denote the speaker related variability. U and V are termed the eigen-channel and eigen-voices respectively and typically $Nd \times 400$ dimensions. D provides a mechanism to capture the residual variability not captured by U and V. Thus, the key idea in the JFA technique is to find two subspaces (V and U) that best capture the speaker and the channel variabilities in the feature space. The term joint factor analysis comes from the fact the three latent variables x, y and z are jointly estimated unlike traditional factor analysis where an independent estimation is adopted.

Dehak et al. [24] observed that the channel subspace still contains information about the speaker and vice-versa. Therefore, a combined subspace was proposed to capture both variabilities and called the *total variability space*. In this formulation, the speaker- and channel-dependent supervector s is modeled as

$$s = m + Tw, \tag{8.3}$$

where m is a speaker- and channel-independent supervector (usually the UBM supervector), T is a low-rank $Nd \times 400$ dimensional matrix representing the basis of the total variability space, and w is a normal-distributed vector representing the coordinates of the speaker in that space. The vector w is called the *i-vector*, short for "intermediate vectors" due to their intermediate representation between the acoustic and supervector representation; or the "identity vectors" for their compact repre-

sentation of a speaker's identity. The set $\{m, T\}$ represent the hyper-parameters of the total-variability framework. Typically, the number of dimensions of w is three orders of magnitude smaller than that of the supervectors (e.g. 400 vs 10^5). The i-vectors thus provide a concise representation of the high-dimensional supervectors.

8.1.1 Hyper-parameter training

A key difference between the training of total variability matrix T and the eigen-voices V is that for V all utterances from a specific speaker are considered to belong to the same person, whereas T-training takes each utterance to be independent irrespective of the speaker identity. Otherwise, both training takes a similar EM-approach [47]. The eigen-channel U is estimated after estimating the eigen-voices and the approach is detailed in [33]. Both JFA and i-vectors are adept at modeling the intrinsic variabilities, however, i-vectors are increasingly being preferred due to their superior recognition performance and compact representation.

8.1.2 Intersession compensation in i-vector space

The i-vector representation does not include explicit compensation for the intersession variabilities like in JFA. However, standard intersession compensations have been proposed in the i-vector space [25]. The most successful compensation include a Linear Discriminant Analysis (LDA) projection followed by Within Class Covariance Normalization (WCCN).

Linear discriminant analysis (LDA) looks to find orthogonal directions that

simultaneously maximize the inter-speaker discrimination and minimize the intra-speaker variability. This is analogous to learning a direction that removes channel and other nuisance directions from the i-vectors. The idea behind WCCN is to scale the total variability space by a factor that is inversely proportional to an estimate of the within-class covariance. This deemphasizes the directions of high intra-speaker variability.

A LDA based subspace is first learnt on the i-vectors, and training and testing i-vectors are projected into this space. Let L denote the LDA projection matrix. Then, a within-class covariance normalization matrix W is learnt on LDA-projected space. A compensated i-vector for a raw i-vector w is given by,

$$\hat{w} = W^{-0.5}(Lw). \tag{8.4}$$

A detailed description of the nuisance compensation in i-vector space is available in [25]. Given the compensated i-vectors, the key challenge in speaker recognition is the design of appropriate learning techniques that can classify speakers in the i-vector space.

8.2 Kernel Partial least squares (KPLS)

A brief description of kernel PLS is provided here; more detailed analysis is available in [76]. Denote a d-dimensional feature by x (the i-vector from a single speech utterance in our case) and the corresponding speaker label by y. KPLS considers the mapping of the features x to a higher dimensional space, given by $\Phi : R^d \Rightarrow R^{\bar{d}}$. Assume momentarily that such a Φ is defined and known. KPLS

formulation proceeds similar to the linear counterpart in Section 7.2, by replacing X with $\Phi(X)$. Similar to the linear PLS, KPLS is also solved via the *nonlinear iterative partial least squares (NIPALS) algorithm* (Alg. 5) [75], which constructs a set of weight vectors $W = \{w_1, w_2, \ldots, w_p\}$ such that

$$\max[\text{cov}(t_i, u_i)]^2 = \max_{|w_i|=1} [\text{cov}(\Phi(X)w_i, Y)]^2, \tag{8.5}$$

where t_i and u_i are the i^{th} columns of T and U respectively and $\text{cov}(t_i, u_i)$ indicates the sample covariance between latent vectors t_i and u_i. Maximizing the covariance in the latent vector space is equivalent to maximizing discrimination in the same space; in other words, for a particular speaker, KPLS learns a subspace in which the speaker latent vectors t_S are well separated from the imposter latent vector t_I as illustrated in Figure 8.1. Thus, KPLS learns a unique latent space for each speaker.

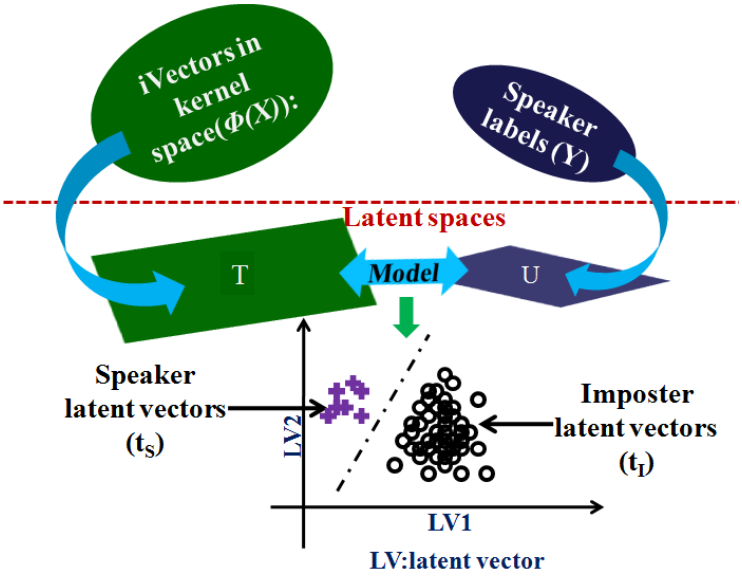


Figure 8.1: [color] *Non-linear mapping and the corresponding subspaces learnt via Kernel Partial Least Squares (KPLS).*

It has been shown [75] that the NIPALS algorithm is equivalent to iteratively

finding the dominant eigenvectors of the problem

$$[\Phi(X)^T yy^T \Phi(X)]w_i = \lambda w. \tag{8.6}$$

The $\Phi(X)$-scores t_i are then obtained as $t_i = \Phi(X)w_i$. Rosipal et al. [76] modify this eigenproblem as

$$[\Phi(X)\Phi(X)^T yy^T]t = \gamma t. \tag{8.7}$$

Using the "kernel" trick [7], $\Phi(X)\Phi(X)^T$ can be defined as a kernel matrix K leading to the final eigenproblem

$$[Kyy^T]t = \gamma t. \tag{8.8}$$

A key advantage of this kernelization is that an explicit definition of the mapping function Φ is not required and it suffices to define a kernel function between pairs of feature vectors. This modified version of the NIPALS algorithm for KPLS has been detailed in [76].

After extraction of latent vectors t_i and u_i, the kernel matrix K is deflated by removing any information captured by t_i and u_i from K:

$$K \leftarrow (I_n - tt^T)K(I_n - tt^T). \tag{8.9}$$

The process is repeated till a sufficient number (determined via standard cross-validation) of latent vectors is obtained.

8.2.1 KPLS speaker models

We use the KPLS in the regression framework [104] for speaker modeling. Substituting the w from Eq. (7.4) in Eq. (7.2), we get

$$\Phi(X)W = TP^TW + E \Rightarrow T = \Phi(X)W(P^TW)^{-1}. \tag{8.10}$$

Now, U can be written in terms of T as $U = TD + H$, where D is a diagonal matrix and H is the residual [75]. Eq. (7.3) now becomes

$$Y = TDQ^T + HQ^T + F = \Phi(X)W(P^TW)^{-1}DQ^T + \bar{F}.$$

Using $P = \Phi(X)^TT$ and $W = \Phi(X)^TU$ from [76] in the above equation, we generate the score for a test i-vector:

$$\text{score}_{\text{KPLS}} = \Phi(X_t)\Phi(X)^TU(T^T\Phi(X)\Phi(X)^TU)^{-1}DQ^T.$$

This leads to the KPLS regression:

$$\text{score}_{\text{KPLS}} = K_tB; \quad B = U(T^TKU)^{-1}DQ^T, \tag{8.11}$$

where B is the set of PLS regression coefficients, K_t is the kernel matrix between training data and testing data i-vectors, and K is the kernel matrix between the training data i-vectors only. The regression framework provides a direct method to compute the "KPLS speaker models". The kernel matrix deflation requires the explicit construction of the kernel matrix, and the accelerated construction from GPUML was used for this purpose.

The KPLS score is obtained using Eq. (8.11) with the kernel built using testing data i-vectors against the development data and training data i-vectors. *Note that*

the KPLS score is a linear combination of the cosine scores between the testing data i-vector and the combination of target data and development data i-vectors and that this linear combination (B) is unique to each speaker.

8.3 One-shot similarity scoring

One shot similarity (OSS) draws its motivation from the class of one-shot learning techniques that learn from one or few training examples [117]. It has been explored in the contexts of face recognition [116] and insect species identification [117]. OSS compares two vectors by considering a *single, unlabeled negative example set* and using it to learn what signals are considered "different". Given a definition for a set of background data (negative examples) A , one shot similarity compares any pairs of feature-vectors x and y to provide their similarity scoring. OSS first computes a model for the vector x and then uses the model to score y. Intuitively, this score would give the likelihood of y belonging to the same class as x. A similar score for x based on a model built with y is obtained. The one-shot similarity (OSS) score is then obtained by averaging these two scores. These steps are illustrated in Fig. 8.2.

In the context of speaker recognition and KPLS, the single, unlabeled negative example set A is the set of i-vectors from background speakers, x is the target speaker's training i-vector and y is the test i-vector. We use KPLS to model each vector based on the negative example set.

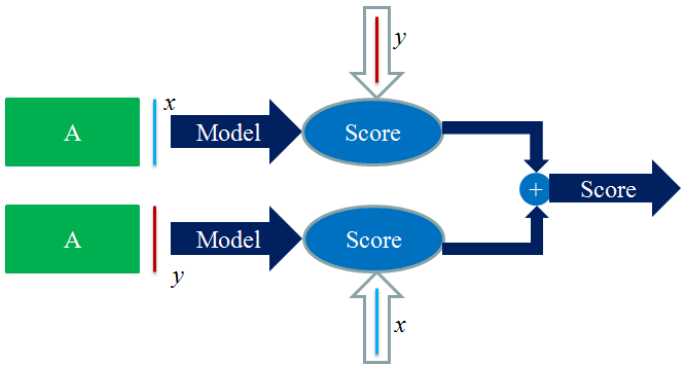


Figure 8.2: [color] *One Shot Similarity scoring*

8.3.1 Present approach:

In our experiments, we use gender dependent sets A. For each target speaker, the corresponding i-vector is assigned a label of $+1$; samples in set A are assigned a label of -1; the KPLS model (one-vs-all approach) is trained; and the speaker-specific regression coefficients B are learnt according to Eq. (8.11). This is repeated for both train and test i-vectors. KPLS output scores in each case are Z-normalized [5]. The scores are then combined to a one-shot similarity score. These steps are summarized in Figure 8.3.

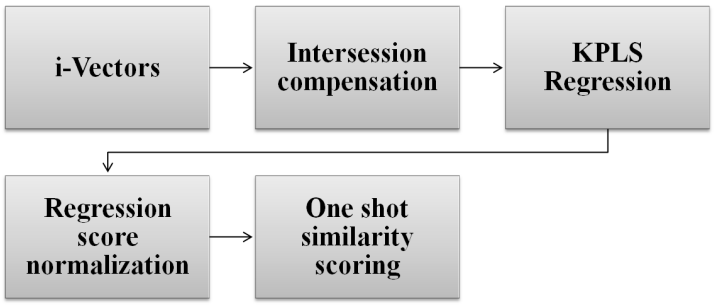


Figure 8.3: *One-shot schematic for speaker recognition.*

8.4 Experiments

We performed experimental evaluation of the proposed method on the *extended core set* of the NIST SRE 2010 evaluation data set, which is grouped into 9 trial conditions[1]. Our development data consisted of NIST SRE 2004, 2005, 2006, and 2008 data; Switchboard data set, phases 2 and 3; Switchboard-Cellular data set, parts 1 and 2; and Fisher data set (total of 17319 male and 22256 female utterances). A gender-dependent 2048-center UBM with diagonal covariance was trained using the standard 57 MFCC features, and the gender-dependent total variability matrix T of dimension 400 was also learnt.

8.4.1 Parameters of KPLS/OSS

Two main parameters of the KPLS/OSS system are the kernel function and the set A in one-shot framework.

8.4.1.1 Choice of kernel

There are several popular kernels used with kernel methods. We explored two popular kernels namely Gaussian (Eq. 8.12) and Cosine (Eq. 8.13).

$$k_{\text{Gaussian}}(\hat{w}_1, \hat{w}_2) = \exp\left[\frac{\|\hat{w}_1 - \hat{w}_2\|^2}{2}\right] \tag{8.12}$$

$$k_{\text{Cosine}}(\hat{w}_1, \hat{w}_2) = \frac{\left(\hat{w}_1^T \hat{w}_2\right)}{\left(\hat{w}_1^T \hat{w}_1\right)\left(\hat{w}_2^T \hat{w}_2\right)} \tag{8.13}$$

The performance of KPLS on the SRE 2010 extended core dataset based on these two kernels is shown in Fig. 8.4. Both these kernels have comparable results, the cosine

[1] `www.itl.nist.gov/iad/mig/tests/sre/2010/`

kernel has a marginally better performance than Gaussian in several conditions and hence was used in our subsequent experiments.

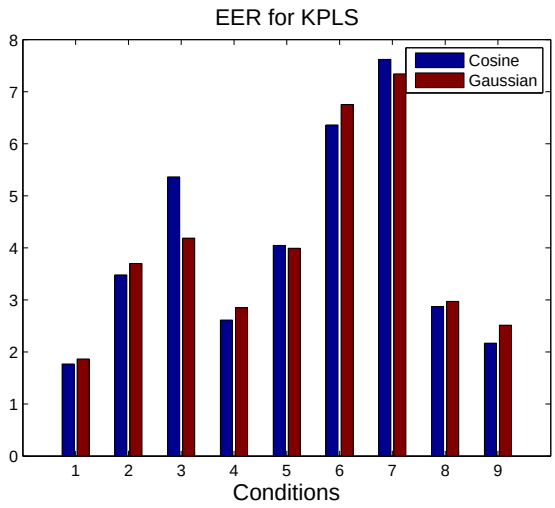


Figure 8.4: [color] *Performance of KPLS on the SRE* 2010 *extended core dataset based on the Gaussian and Cosine kernels*

8.4.1.2 Set of negative examples A

The background data or the negative example set A determines the distribution of the negative examples. The size of this set is fixed and the samples are chosen from our development data in such a way that the resulting speaker contains the maximum possible distinct encompassing various channels in the combined databases. Such a choice would enable the best possible discriminative framework. The effect of the size of the set A is shown in Fig. 8.5 where the EER for condition 2 (interview speech from different microphone for training and testing) in SRE 2010 extended core task is shown for various sizes of the set A.

The drop in EER from size $\sim$ 1000 to a size $\sim$ 5000 is considerable (4.2 $\rightarrow$

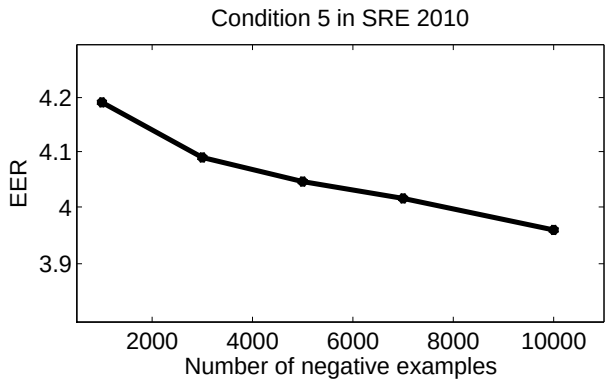


Figure 8.5: [color] *Effect of the size of the set of negative examples A in the performance: EER for condition* 2 *in the SRE* 2010 *extended core task with OSS-KPLS*

4.0). However, the drop from ~ 5000 to ~ 10000 is not equally significant ($4.0 \rightarrow 3.95$), considering the increased computational cost associated with the increase. Therefore, the size of A was set at ~ 5000 in our experiments.

8.4.2 Systems compared

The proposed OSS-KPLS based speaker recognition was compared against several state-of-the-art systems, specifically JFA [10] [47], PLDA [45], and CDS [23]. We describe these systems briefly here.

8.4.2.1 Joint Factor Analysis

JFA provides an explicit mechanism to model the undesired variability in the speech signal. It decomposes the speaker supervector as

$$s = m + Ux + Vy + Dz, \tag{8.14}$$

where $\{m, U, V, D\}$ are the hyper-parameters of the JFA model, which are estimated via Expectation Maximization (EM). The rationale behind this equation is that aside from the offset m (UBM), the mean supervector is the superposition of three fundamental components with rather distinctive meaning. The component that lives in U is used to denote undesired variabilities in the observed vectors (e.g. channel related variability). The component living in V is used to denote the speaker related variability. D provides a mechanism to capture the residual variability not captured by U and V.

Defining $\Phi_{JFA} = [UVD]$ and $\beta = [xyz]^T$, we obtain compensated training and testing supervectors as

$$\eta_{train/test} = \Phi_{JFA}\beta - Ux_{train/test}. \tag{8.15}$$

The final JFA score is then given by

$$\text{score}_{\text{JFA}} = \frac{1}{N}\eta_{train} W_{test} \eta_{test}; \tag{8.16}$$

where N is the number of frames in the test segment, $W_{test} = \Gamma_{test}\Sigma^{-1}$, Γ_{test} is the soft-count of each Gaussian mixture as defined in [33] and Σ is the variance of the UBM.

In our experiments, we use the JFA as described in [33]. The U and V matrices are learnt with 300 and 100 dimensions respectively. The final JFA scores are then ZT-normalized [5].

8.4.2.2 Probabilistic Linear Discriminant Analysis

PLDA facilitates the comparison of i-vectors in a verification trial. A special *two-covariance* PLDA model is generally used for speaker recognition in the i-vector space. The speaker variability and session variability are modeled using across-class and within-class covariance matrices (Σ_{ac} and Σ_{wc} respectively) in the PLDA setup. A latent vector y representing the speakers is assumed to be normally distributed $\mathcal{N}(y; \mu, \Sigma_{ac})$, and for a given speaker represented by this latent vector, the i-vector distribution is assumed to be $p(w|y) = \mathcal{N}(w; y, \Sigma_{wc})$.

Given two i-vectors w_1 and w_2, PLDA defines two hypotheses $\mathcal{H}_s$ and $\mathcal{H}_d$ indicating that they belong to the same speaker or to different speakers respectively. The score is then defined as $\log \frac{p(w_1,w_2|\mathcal{H}_s)}{p(w_1,w_2|\mathcal{H}_d)}$. Marginalization of the two distributions with respect to the latent vectors leads to

$$\text{score}_{\text{PLDA}} = \log \frac{\mathcal{N}\left(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}; \begin{bmatrix} \mu \\ \mu \end{bmatrix}, \begin{bmatrix} \Sigma_{tot}\Sigma_{ac} \\ \Sigma_{ac}\Sigma_{tot} \end{bmatrix}\right)}{\mathcal{N}\left(\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}; \begin{bmatrix} \mu \\ \mu \end{bmatrix}, \begin{bmatrix} \Sigma_{tot}0 \\ 0\Sigma_{tot} \end{bmatrix}\right)}$$

In our experiments, we found that using a Σ_{ac} of rank 200 along with a full-rank (rank 400) matrix Σ_{wc} produced the best results. The scores were S-normalized only for those conditions that involve telephone speech (all except C1, C2 and C4, where S-norm was found to be detrimental for both EER and DCF). The S-norm is defined in [45] and can be interpreted loosely as a symmetric version of Z-norm [5].

8.4.2.3 Cosine Distance Scoring:

The CDS classifier has been used by Dehak et al. [24] and Senoussaoui et al. [84]. Improved performance has been reported over the corresponding SVM-based approach. The CDS classifier defines the score for the trial as a cosine similarity function between two i-vectors after projecting them to an LDA subspace (learnt on the development data) to remove the session variability. If w_1 and w_2 are the training data and the testing data i-vectors and A is the LDA projection matrix, the CDS score is given by

$$\text{score}_{\text{CDS}} = \frac{(Aw_1)^T(Aw_2)}{\sqrt{(Aw_1)^T(Aw_1)}\sqrt{(Aw_2)^T(Aw_2)}}. \tag{8.17}$$

In out experiments, the CDS scores were Z-normalized [5].

8.4.3 Results

We compared the performance of the OSS-KPLS based speaker recognition against the JFA, PLDA, CDS and KPLS systems. The corresponding equal error rate (EER) and detection cost function (DCF) values across each condition are tabulated in Table 8.1 and are shown graphically in Figure 8.6. The DCF is defined as for NIST SRE 2010 "core" and "8conv/core" conditions. The corresponding DET curves are shown in Fig. 8.7.

The PLDA and JFA systems belong to the class of generative methods for speaker recognition. Between them, the PLDA is better in most of the conditions. In contrast, OSS-KPLS and CDS belong to the class of discriminative methods, and OSS-KPLS outperforms CDS in most of the conditions (in terms of EER).

Systems	C1	C2	C3	C4	C5	C6	C7	C8	C9
JFA	2.67	4.34	4.06	3.65	3.55	7.04	8.16	3.11	2.13
	0.53	0.62	**0.55**	0.63	0.52	0.86	0.96	**0.50**	0.43
PLDA	1.77	**3.09**	**3.00**	2.85	**2.59**	**5.43**	8.06	**2.51**	2.17
	0.36	0.59	0.56	0.50	**0.49**	**0.79**	0.86	0.52	0.39
CDS	2.27	4.06	3.71	3.48	4.18	6.36	8.46	2.99	1.96
	0.33	0.55	0.58	0.47	0.56	0.82	**0.76**	0.52	**0.32**
KPLS	1.77	3.48	5.36	**2.61**	4.05	6.36	7.62	2.87	2.17
	0.33	0.57	0.62	0.47	0.61	0.85	0.85	0.55	0.34
OSS	**1.65**	3.57	4.84	2.85	3.84	6.19	**7.34**	2.73	**1.82**
	0.31	**0.55**	0.59	**0.45**	0.55	0.84	0.82	0.53	0.33
OSS+	**1.58**	**2.76**	**2.97**	**2.48**	**2.59**	**5.39**	**7.16**	**2.40**	**1.61**
PLDA	0.35	0.59	0.56	0.50	**0.49**	0.79	0.86	0.51	0.36

Table 8.1: *Equal error rate (EER) and detection cost function (DCF) values obtained using Joint Factor Analysis, Probabilistic Linear Discriminant Analysis, Cosine Discriminative Scoring, Kernel Partial Least Squares and One-shot/KPLS classifiers for the NIST SRE 2010 extended core data set.*

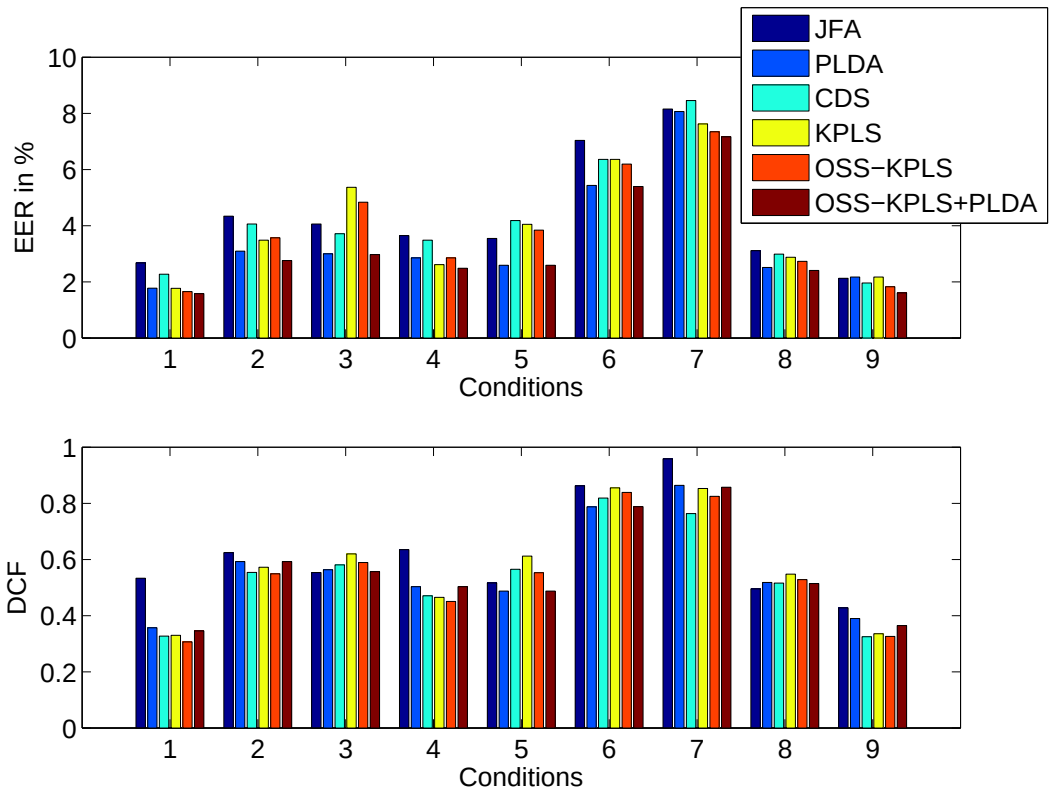


Figure 8.6: [color] *Performance of JFA, PLDA, CDS, KPLS and OSS-KPLS classifiers on the NIST SRE 2010 extended core data set.*

OSS-KPLS performance was comparable to PLDA performance in several testing conditions in terms of EERs and DCFs and is better than PLDA in 3 of the 9 conditions for both EER and DCF. Also note that in most conditions OSS-KPLS is easily the second best system (if not the best) in terms of both EERs and DCFs for most of the tested conditions.

Given that the PLDA and OSS-KPLS perform consistently better than other systems, we explored the possibility of score fusion between these approaches. We computed the fused score by combining the output scores with linear weights, which were trained using a small subset of development data. The results are also shown

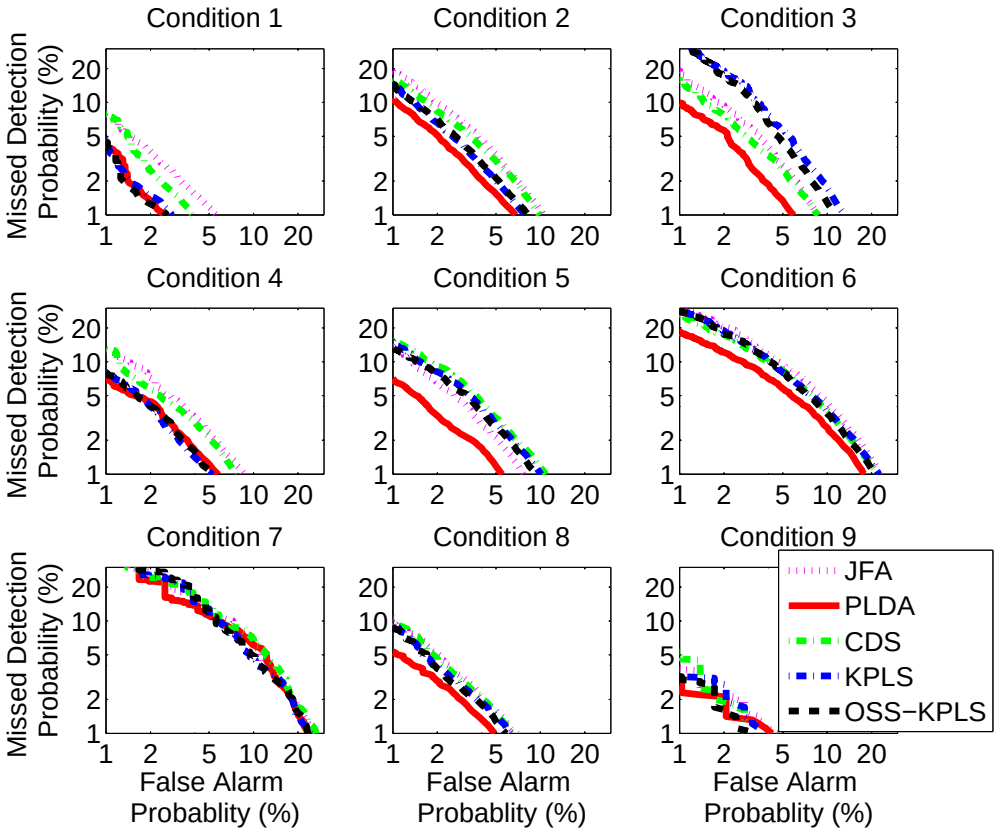


Figure 8.7: [color] *Performance of JFA, PLDA, CDS, KPLS and OSS-KPLS classifiers on the NIST SRE 2010 extended core data set: DET curves for various test conditions*

in Table 7.1 and Figure 7.6. The fused scores yield the best EERs in all conditions, suggesting the complementary nature of PLDA and OSS-KPLS in capturing speaker characteristics. More sophisticated fusion strategy is a subject of further research.

8.4.4 Effect of Noise

In order to test the noise sensitivity of the OSS-KPLS system, babble noise of various levels were added to all test utterances and the individual systems' perfor-

mances were evaluated. The results for Condition 2 (interview speech from different microphone for training and testing) of the SRE 2010 extended core is shown in Fig. 8.8. It can be seen that additive noise deteriorates the performance of all the systems. However, OSS-KPLS is the second best system even in the presence of noise in terms of EERs and is the best system in terms of DCFs.

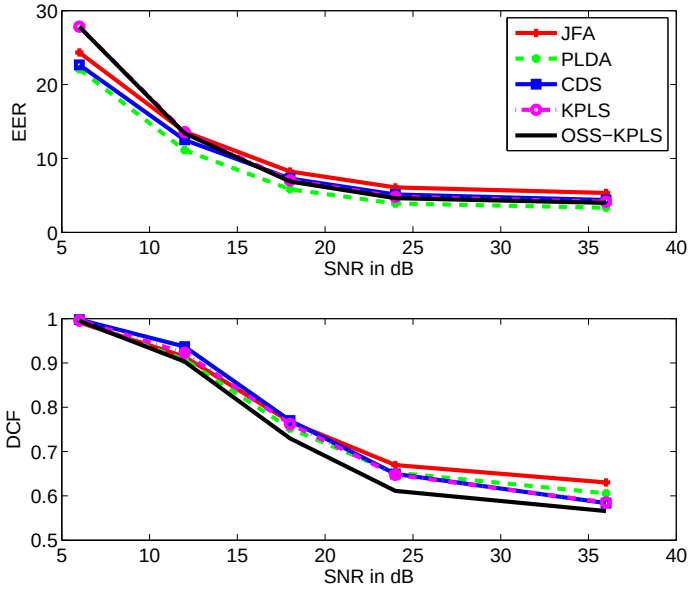


Figure 8.8: [color] *Sensitivity of JFA, PLDA, CDS, KPLS and OSS-KPLS to additive babble noise on the Condition* 2 *of SRE* 2010 *extended core dataset*

8.5 Conclusions

In this chapter, we have extended kernel partial least square into a one-shot-similar framework for speaker recognition. The one-shot similarity enables a symmetric scoring for the KPLS-based discriminative framework. The proposed framework was compared against several state-of-the-art systems on the NIST SRE 2010

extended core data set. The OSS-KPLS system outperforms the state-of-the-art in several conditions and provides complementary information, resulting in further improved performance using simple linear score combination as a score fusion technique. Performance analysis in the presence of noise indicates that the new system maintains the performance gap even in the presence of noise.

Chapter 9

PLS for loan defaults prediction

We have earlier seen the application of partial least squares for the problem of speaker recognition in Chapter 7. While our focus there was on speaker recognition, the method itself is generic and has several practical applications. In this chapter, we explore one such application to a totally different domain: risk prediction with student loans.

9.1 Loan monitoring and warning systems

Several financial institutions proactively monitor loans of various types (by lender, term, age, default criterion etc.) and from several domains (such as auto, home, student etc.). When the monitoring relies on a manual process and the number of active loans is enormous, it is challenging to predict the total number of loans that have a high propensity to not pay back and which those that are at a high risk of defaulting in the near future. Without the aid of an automated system, it would be a lengthy, erroneous, and/or one-size-fits-all type montioring process. However, if the risk associated with a given loan record can be predicted well ahead of time, the agency can carry out appropriate communication or other suitable action to prevent loan default.

For a given portfolio, variables at various scales can be observed or deduced.

On a personal level, it is possible to observe the borrower-specific payment patterns such as loan repayment behavior (current, delinquent, forbearances used, etc), inter-payment gaps, and the rate of communication. Various interactions of a borrower - as with the call center servicing the customer and online portal activity regarding the loan can be recorded. It is also possible to include regional scale data such as the unemployment rate that would signify the general repayment capacity of the regional population. Observations can also be aggregated at the national scale from national institutions like Department of Labor, etc. In this work, we use data from an actual loan services provider and the exact details of the agency and the observations are deliberately omitted due to privacy constraints.

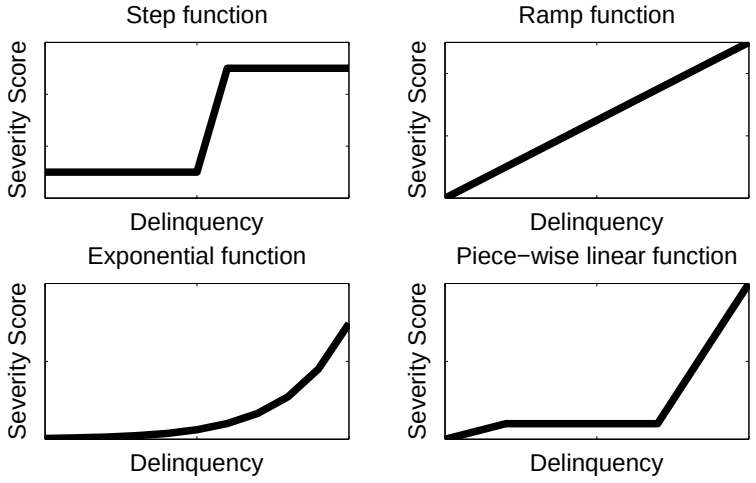


Figure 9.1: *Possible mapping of risk severity*

In order to predict the risk associated with a loan, it is required to define a quantitative mapping of observed variables to risk that can be used by an agent to take appropriate decision. A domain expert can provide pointers to functional mappings such as a ramp, piecewise-linear, exponential, etc [Fig. 9.1]. We use a

combination of these functions to map the delinquency to a risk severity of a loan in a scale of 0 to 100. Again, due to proprietary reasons, the exact nature of the risk function is not revealed here.

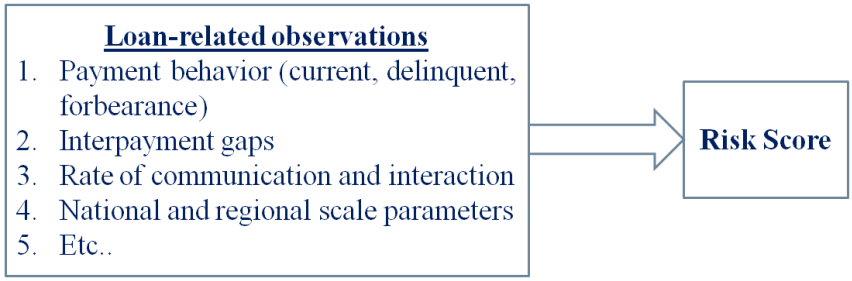


Figure 9.2: *Loan related observations are used to model the associated risk*

Given the observations and a quantitative risk, we formulate the task of risk severity prediction as a regression problem [Fig. 9.2]. We extend the partial least squares regression[75] from Chapter 7 to model the risk severity and use the associated variable influence on projection to choose the most influential variables for prediction. We tune the overall approach to suit the nature of this problem, the key challenges here are,

1. A very small part of an observed population constitutes defaulted case that is required to be modeled. Any model must address this imbalance

2. The financial status of a person can be very volatile, in order to capture such volatility special indicator variables on top of the model predictions.

3. To handle the large data sizes, the method must be computationally scalable.

Observations over a 12-month period are used to model the the loan severity 6 months ahead of time (risk at the 18th month). The model capabilities are tested

on an independent set of records which has approximately the equal same size as the training data.

This chapter is organized as follows. We introduce the Variable influence of projection (VIP) based variable selection in Section 9.2. Domain specific adaptation of PLS and VIP to improve the predictions is discussed with our experiments in Section 9.3 and Section 9.4 concludes the chapter.

9.2 Variable influence on projection

Although traditional regression models will be able to deal with a large number of highly correlated variables (predictors or descriptors), there are several situations in which better predictions are obtained when a subset from a larger number of variables is selected. This occurs mainly because in a set of hundreds of variables, most of them enclose noise, irrelevant and/or redundant information. Feature selection is a way to identify variable subsets that in fact reproduce the observed values of a dependent variable, i.e. those subsets that are, for a proposed problem, the most useful to obtain a more accurate regression model. Although the main emphasis in variable selection is upon the prediction, it is desirable that the selected subsets should aid the interpretation of the regression model. Thus, the aim of variable selection is to reduce significantly the number of variables to obtain simple, robust and interpretable models[106]. Here, we use the Variable Influence on Projection or VIP to select the most important variables for prediction.

The VIP score of a predictor, first published by Wold et al. [115], is a summary

of the importance for the projections to find f latent variables. The VIP score for the j-th variable is a measure based on the weighted PLS coefficients W (from Eq. 7.4). Given W and the latent vectors T and U, the VIP score for the j-th variable is given by,

$$\begin{aligned} \mathrm{VIP_j} &= \sqrt{\frac{\sum_f b_f^2 W_{jf}^2}{\sum_f b_f^2}}; \\ \text{where} \quad \mathrm{b_f} &= \sum_f \frac{t_f^T u_f}{t_f^T t_f}. \end{aligned} \tag{9.1}$$

Once the VIP scores are computed, an appropriate "threshold" can be used to choose the variables. However, since the average of squared VIP scores equals 1, the *"greater than one rule"* is generally used as a criterion for variable selection [18].

9.2.1 PLS model for loan prediction

In our PLS model for loan prediction, we first determine the influential variable via VIP. Once the variables are selected, the regression coefficients are learnt on this reduced set. A 10−fold cross-validation is used to select the number of latent PLS factors in each case.

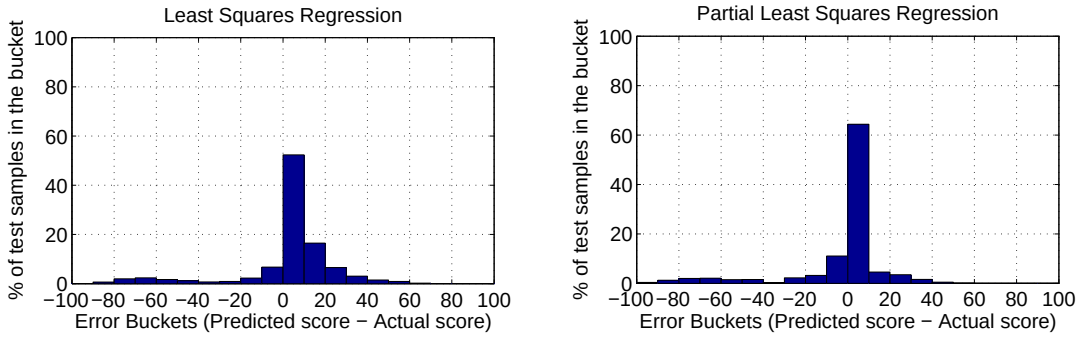
9.3 Experiments

In our experiments, we first illustrate the performance of PLS regression against a standard least squares regression. Due to its better performance PLS was chosen for further model improvements. We address the data imbalance via multiple PLS models and the volatility in data by using indicator variables. The

PLS approach is computationally efficient and the scalability has already been addressed in Chapter 7.

9.3.1 Least Squares Regression vs PLS Regression

In our first experiment, we compared the PLS regression with a least squares regression. The variable selection in PLS was performed with VIP-scores, with the least squares regression, variable selection was performed using the method in [32]. As a first measure of comparison, we look at the histogram of the error for the two approaches in Fig. 9.3. For the least squares regression, around 60% of the predicted scores fall within an error range of ±10, whereas close to 75% of the PLS-predicted scores fall within ±10 range. Also, the computation complexity of PLS regression is better than the least squares regression due to the expensive pseudo-inverse computation in least squares regression. Due to these reasons, we use PLS for our further analysis. However all the improvements we propose are applicable to the least squares regression as well.



(a) Least squares regression

(b) PLS regression

Figure 9.3: [color] *Error histogram: Least squares regression vs PLS*

The confusion matrix of the baseline PLS-model is shown in Table 9.1. It can be seen that there is low accuracy in high risk predictions. A similar distribution is also observed with the least squares regression as well requiring further modifications of the regression model to handle the problem.

		Predictions		
		Low Risk	Medium Risk	High Risk
Ground truth	Low Risk	84.00	2.03	0.00
	Medium Risk	5.25	1.01	0.00
	High Risk	5.30	2.41	0.00

Table 9.1: *Confusion matrix for the PLS regression model, numbers shown in percentages of the total records*

9.3.2 Subpopulation modeling

For the domain that we are building the model, it is necessary to have higher accuracy at the high risk region even if this is at the expense of a lower accuracy at the low risk region. One possible reason for the low-accuracy-high-risk prediction is the the data imbalance. To address this, we propose to build separate models to different subpopulation of the data; this idea is summarized in Fig. 9.4. The subpopulation can be selected based on aggregated observed values or some indicators derived from the observations.

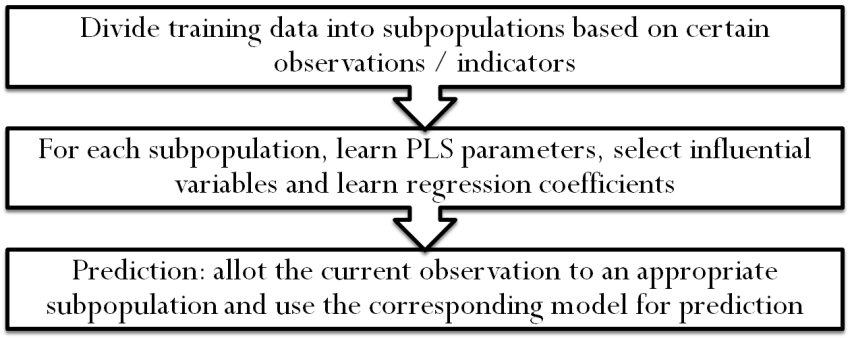


Figure 9.4: *Subpopulation model to improve data imbalance*

9.3.2.1 Observation-based subpopulation

One way to select a subpopulation is by clustering the average of a particular variable in the observation period. In our experiment, we averaged the quantized risk in the observed period and divided that into three clusters of low/medium/high risk. Such a segregation helps overcome the data imbalance to some extent as indicated by the corresponding confusion matrix in Table. 9.2.

		Predictions		
		Low Risk	Medium Risk	High Risk
Ground truth	Low Risk	81.41	4.33	0.27
	Medium Risk	3.90	2.26	0.10
	High Risk	3.41	3.67	0.63

Table 9.2: *Confusion matrix for the subpopulation based multi-PLS regression model - subpopulation selected based on aggregated observed risk, numbers shown in percentages of the total records*

9.3.2.2 Indicators-based subpopulation

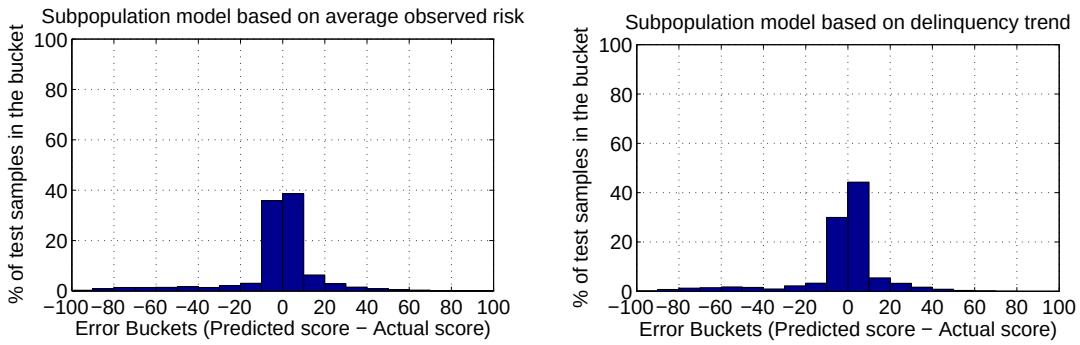
In the last experiment, the subpopulation was chosen solely based on the aggregate of an observed value. An alternate is to do some manipulations on certain observed variables leading to an indicator and segregating the population based on this indicator. In this experiment, we compute the slope of the delinquent days in the last financial quarter and use it as an indicator to segregate the population. Such a segregation also provides similar improvements like the last experiment and the confusion matrix is shown in Table. 9.3.

		Predictions		
		Low Risk	Medium Risk	High Risk
Ground truth	Low Risk	81.40	4.34	0.27
	Medium Risk	4.36	1.79	0.11
	High Risk	3.38	3.77	0.57

Table 9.3: *Confusion matrix for the subpopulation based multi-PLS regression model - subpopulation selected based on the slope of delinquent days (indicator variable), numbers shown in percentages of the total records*

Both the indicator-based subpopulation and aggregated-observation based subpopulation yields comparable results which is also indicated by the error-histograms in Fig. 9.5. We therefore proceed with the aggregated-observation based subpopulation modeling for our further analysis.

While sub-population modeling improves over the single model discussed be-



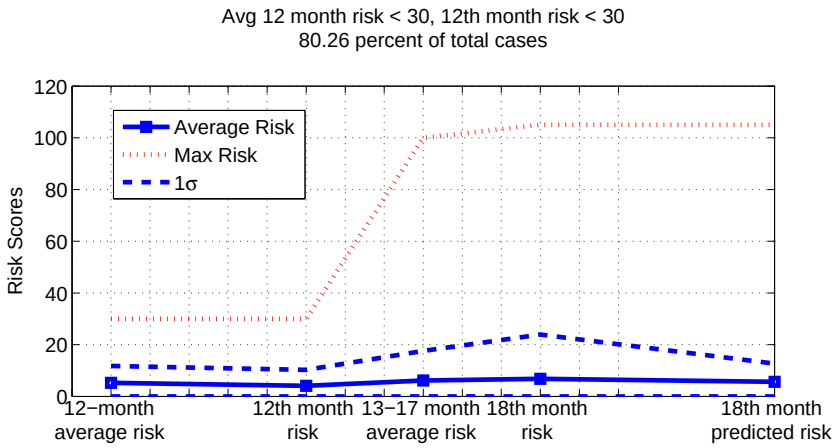
(a) Aggregated risk based subpopulation (b) Slope of delinquency days based subpopulation

Figure 9.5: [color] *Error histogram for PLS subpopulation models: aggregated observations vs derived indicators*

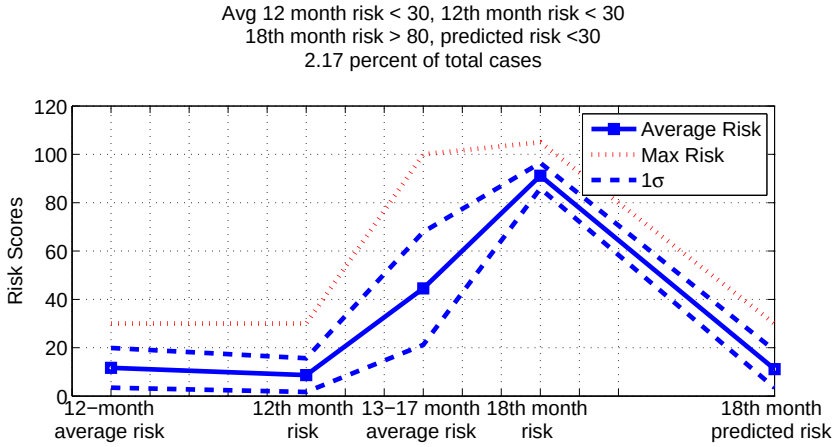
fore, there is still room for improvement. In order to first analyze whether this is due to the model deficiency or any other data related issue, we look at the risks in the observed month, during the unobserved period, predicted risk and the actual risk at the 18-th month. Fig. 9.6(a) shows the general trend all the records where the risk of the borrower/loan is low till the 12-th month and Fig. 9.6(b) shows a subset of these cases where there is a high risk in the 18-th month and the model fails to catch these. These results illustrate that atleast two-thirds of the high risk records predicted low are due to the volatility in the borrower's risk rather than the model deficiency itself. To address this, we propose to artificially boost the model outputs based on certain inputs.

9.3.3 Indicator variables based boosting

Indicator variables summarize certain special characteristics of a given set of observations. The exact definition of the indicator variables varies from domain to



(a) Low risk prediction: general statistic



(b) Low risk prediction: high risk cases predicted as low risk

Figure 9.6: [color] *Error histogram based on the subpopulation PLS models*

domain. We have already seen the performance of the indicator-based subpopulation model in the last section. In this experiment, we use the indicators to boost the model outputs. For boosting model outputs, we propose a couple of new indicator variables that can signify certain loan/borrower related characteristics:

9.3.3.1 Excursions

We define excursion as the number of times an observed variable exceeds its own mean (or any other level like mean + one-standard-deviation). This has the potential to indicate the frequency of the activity, and was particularly useful for communication related observations.

9.3.3.2 Trend Ratio

We define trend as the slope of a variable of interest in a three-month window. Trend ratio in an observed period is the percentage of the positive trend evaluated in the overlapping time windows of the observed period. This has the potential to indicate whether a particular variable has an increasing trend or a decreasing trend.

While these indicators can be added to the variable set and used for building new models, this will have very little impact on the model due to the quantity of these indicators in comparison to the actual observed variables. We therefore boost the model outputs based on these indicators.

For example, a good borrower would have frequent loan-related interactions with the agency via phone or web , which will be indicated by a higher value of the corresponding excursions. A defaulter on the other hand would have very minimal interactions. Similarly a larger delinquency trend ratio would indicate a higher loan risk. Using rules of these kinds (the thresholds set based on the training data), we boost the model outputs and the corresponding confusion matrix is shown in Table 9.4. There is huge improvement over the subpopulation models for the high-risk

prediction, however this comes at a drop in the accuracy of the low risk predictions. In other words, the model with these modification is very conservative, with high false negatives at the expense of lower false positives; this is also indicated by the error histogram in Fig. 9.7, where the original error distribution is more spread in the new predictions.

		Predictions		
		Low Risk	Medium Risk	High Risk
Ground truth	Low Risk	68.51	16.83	3.39
	Medium Risk	1.19	1.81	0.55
	High Risk	1.24	3.21	3.28

Table 9.4: *Confusion matrix for the subpopulation based multi-PLS regression model based on the approach in Fig. 9.4, numbers shown in percentages of the total records*

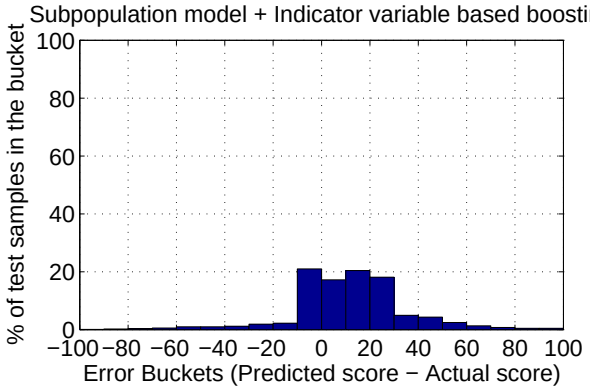


Figure 9.7: [color] *Error histogram on the PLS model: combination of subpopulation based modeling with indicator based boosting*

9.4 Conclusion

In this chapter, we extend the partial least squares regression model to predict the loan severity "n" months ahead of time based on a series of observations ranging from individual behavior to aggregate level statistics. Variable Influence on Projection (VIP) [106] is used to select the most important variables for the prediction. To address the imbalance in the data (large number of low risk records), we proposed to use multiple PLS-models based on subpopulations of the data chosen either via aggregated observations or via derived indicators. To further enhance the output, we define new domain-specific indicator variables that are used to boost the model output on the basis of conditions observed in the data. This results in enhanced performance of the model, particularly for high risk loan records keeping in mind that our goal was to have a conservative model.

Chapter 10

Conclusions

This thesis introduced several novel learning solutions for geospatial interpolation and speaker recognition. The learning algorithms were scaled via the use of graphical processors to handle large datasets.

We began with the key computational primitive in kernel machines and accelerated them on the graphical processors. The resulting acceleration was illustrated in several problem and the core algorithm is released as GPUML, an open-source under Lesser GPL [95, 101]. We extended GPUML to solve the geostatistical kriging [98] by drawing connections with Gaussian process regression. The fast matrix vector products with iterative conjugate gradient solvers are used to scale the kriging and Gaussian process regression to large problems. However, the convergence of iterative solvers becomes an issue for larger datasets, and was addressed with a fast preconditioner in a flexible Krylov solver [97].

We have used the kernelization idea with GPUML to obtain a non-parametric Rényi entropy based information theoretic distance called the kernelized Rényi distance that has potential applications in subset selection [94] and similarity scoring [99]. KRD similarity scoring for speaker recognition performed well against similar approaches, but does not hold its performance against a state-of-the-art recognition system. This led to the formulation of a partial least squares framework for speaker

recognition [104, 103] that has comparable performance to the state-of-the-art systems. The PLS framework was also accelerated on graphical processors to address scalability [102]. The PLS framework was further kernelized to address the nuisance variabilities in speaker recognition, resulting in a very robust recognition system.

10.1 Open problems

The following are some of the open problems in line with the work in this thesis.

10.1.1 Parallelizing linear summation algorithms:

In Section 2.4.2.1, we compared our implementation with a linear version of Gaussian kernel summation. It was evident that inspite of the speedup obtained by our approach, a linear algorithm will eventually beat it. Motivated by this, we tried to map parts of the linear algorithm in [58] on the GPU and achieved speedups up to 3X for some stages. The chief bottle-neck here is the construction of underlying data-structures and if there can be an appropriate map of the data structures to the GPU, the speed up can be substantially increased like in [36].

10.1.2 Other parallel paradigms

While GPU's power is evident from our results, they are definitely not the panacea for all ranks of computational problems. Several parallel programming models are increasingly coming up to address the large amounts of data encountered

in several applications. Popular ones include Hadoop (map/reduce) and MPI. GPU-like multithreaded programming on CPUs is enabled via OpenMP. In this thesis, GPUs have satisfied our computational requirements in most problems and hence was used. In a practical problem, the choice of a particular paradigm is dictated by the needs of the application and the nature of the underlying data.

Typically, when the data dimensions are smaller (< 100), GPUs provide huge improvement in computational performances (provided the algorithm has a "parallelizable" part). For very large data dimensions, a combination of MPI with OpenMP can yield better performances. Although Hadoop is designed for a special suite of problems, they are being increasingly preferred for many applications due to their large scale parallelization (across several 1000s of nodes in the cloud) and associated fault-tolerance guarantees. It would be interesting to explore the extendability of the learning solutions in this thesis to these paradigms.

10.1.3 Co-kriging

We have proposed new acceleration schemes for kriging and have successfully deployed it to reconstruct missing data in satellite observations. Another aspect that can be considered is to krige the distributed and discontinuous data for meteorological and air quality parameters and health survey data to find correlations between them and causative links to develop predictive models. Our kriging framework can be used with such disparate datasets as well where spatio-temporal accuracy of the data-filling is very critical for analyzing. An interesting direction could be the study

of co-kriging between disparate variables in this context. This requires both model-level questions and scalability-related questions to be answered to come up with the appropriate framework.

10.1.4 Quadratic Rényi entropy between GMM

We have independently looked at GMMs for speaker recognition and the Quadradic Rényi entropy for similarity scoring. An interesting extension is to use the Rényi entropy to evaluate distances between GMMs. Given 2 Gaussian mixture models,

$$\mathcal{G}_1(x) = \sum_{j=1}^{N} \pi_j^1 \mathcal{N}(x|\mu_j^1, \Sigma_j^1), \text{and} \tag{10.1}$$

$$\mathcal{G}_2(x) = \sum_{j=1}^{M} \pi_j^2 \mathcal{N}(x|\mu_j^2, \Sigma_j^2). \tag{10.2}$$

The quadratic Rényi information potential between them can be obtained as,

$$\begin{aligned}
\mathcal{R}(\mathcal{G}_1 \| \mathcal{G}_2) &= -\log \int \mathcal{G}_1(x)\mathcal{G}_2(x)dx, \\
&= -\log \int \sum_{i=1}^{N}\sum_{j=1}^{M} \pi_i^1 \pi_j^2 \mathcal{N}(x|\mu_j^1, \Sigma_j^1)\mathcal{N}(x|\mu_j^2, \Sigma_j^2)dx, \\
&= -\log \sum_{i=1}^{N}\sum_{j=1}^{M} \pi_i^1 \pi_j^2 \mathcal{N}(\mu_j^2|\mu_j^2, \Sigma_j^1 + \Sigma_j^2)
\end{aligned} \tag{10.3}$$

With the Rényi information potential, it is possible to obtain a closed-form solution, a difference from the KL-divergence based distances between GMMs [65]. It will be interesting to extend this idea to a suite of problems like in [65].

10.1.5 Improved speaker recognition

The i-vectors in speaker recognition are very successful and we have effectively utilized it in our KPLS frameworks for speaker recognition. The problem of speaker recognition has been studied for several decades now, but still there are several scopes for new research directions.

The i-vectors capture the variabilities in a linear-space; if this can be extended to learn a non-linear manifold and the associated mappings, a better model for variability can potentially be obtained. This raises several research questions such as the best mapping in the context of variability, best back-end system that can learn from these mapping and others which are subjects for further exploration.

Another key area for potential improvements is the performance in noisy environment. Current systems are impressive under clean conditions, but robustness to noisy training and test conditions are being actively explored. It will be interesting to explore the best combination of front-end and back-end to account for noise related variabilities.

Appendix A

Random sampling for testing accelerated algorithms

There are a number of applications where summation of source kernels at a number of target points need to evaluated. These computations have quadratic complexity ($\mathrm{O}(N^2)$) thus hindering its scalability to large datasets. We have seen one approach to accelerate this problem via the use of graphical processors in Chapter 2. However it is not possible to test these approaches for large datasets due to the large cost of direct evaluations. To overcome this, we propose a random sampling approach using which evaluate the error only at K evaluation points, and extrapolate these characteristics to the whole datasets with guaranteed bounds. We derive the sample size K for a desired accuracy based on Chernoff bounds.

A.1 Chernoff Bounds

Chernoff bound gives the upper tail bound ($Pr\left[X \geq \mu(1+\delta)\right]$) and lower tail bound ($Pr\left[X \leq \mu(1-\delta)\right]$). The upper tail bound is given by,

$$\begin{aligned} Pr\left[X \geq \mu(1+\delta)\right] &= \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} \\ &\leq e^{-\mu\delta^2/3} \end{aligned} \tag{A.1}$$

Similarly, the lower tail bound can be given by,

$$Pr\left[X \leq \mu(1-\delta)\right] \leq e^{-\mu\delta^2/2}. \tag{A.2}$$

Eqs A.1 and A.2 gives the Chernoff bound.

A.2 Sampling Problem

The goal in sampling is to select a subset of the original data at random. Let us analyze the property of the resulting subset, let X_i be the random variable corresponding to the i^{th} sample of the subset such that, X_i is 1 if a desired property is satisfied, 0 otherwise. If K is the size of the subset, then it is desired that,

$$Pr\left[\left|\frac{\sum_i X_i}{K} - \frac{M}{N}\right| \geq \epsilon\right] \leq \eta, \tag{A.3}$$

where M is the number of datapoints satisfying the desired property in the original dataset. Let us denote $\frac{M}{N}$ as p and $\sum_i X_i$ as Y, thus Eq. A.3 can be rewritten as,

$$\begin{aligned} Pr\left[\left|\frac{Y}{K} - p\right| \geq \epsilon\right] &= Pr\left[|Y - Kp| \geq K\epsilon\right] \\ &= Pr\left[Y \geq Kp + K\epsilon\right] \\ &\quad + Pr\left[Y \leq Kp - K\epsilon\right] \end{aligned} \tag{A.4}$$

To summarize, we want to select K sized subset from a large data, such that, if M points of the N total points in the full data have a certain property, the property is preserved by a certain number L points in the subset, such that $\frac{L}{K} = \frac{M}{N}$ with high probability given by Eq. A.4.

A.2.1 Adaptation

Before applying the Chernoff bound here, we adapt this to the problem of testing a summation algorithm. The summation algorithm would give the sum at

evaluation points in an efficient fashion. We want to test the accuracy of the fast algorithm. So we shall sample evaluation points at random K points and would evaluate the sum directly at these points. We would then check the accuracy with respect to the fast algorithm to be tested. We expect the error evaluated at all K points to be below a certain threshold.

Let us assume that the fast algorithm assures an error bound of ς. Let us define the property that we shall look for in the data as the error between the direct and fast approach $\leq \varsigma$. Because the algorithm assures such an error bound, the p in Eq. A.4 is 1. The expected number of the points in the subset that will hold the error property is K, thus $\mu = K$. Applying Chernoff bound and substituting $p = 1$ in Eq. A.4,

$$\begin{aligned} Pr\left[Y \geq K(1+\epsilon)\right] &+ Pr\left[Y \leq K(1-\epsilon)\right] \\ &\leq 0 + e^{K\epsilon^2/2} \\ &\leq e^{K\epsilon^2/2} \leq \delta \\ \Rightarrow K &\geq \frac{2}{\epsilon^2}\log\left(\frac{1}{\delta}\right) \end{aligned} \tag{A.5}$$

Thus, setting the parameters ϵ,δ and ς, we can choose K points uniformly at random from the original data set, evaluate the sum directly and test for the desired error bound. If all the points satisfy the required error bound ς, algorithm can be declared accurate within confidence interval ϵ and probability $1 - \delta$.

In order to validate this bound, we compare the GPU based kernel summation in Chapter 2 for Gaussian kernel and corresponding direct-double-precision version.

We evaluated the sum of $10,000$ Gaussian kernels for $10,000$ points. We use the bound in A.5 and evaluated the kernel at $2,952$ points with the direct approach to test the error. The error was less than 10^{-5} at all these points. Therefore, our bound leads to the suggestion that at least 95% of the samples have a relative error $\leq 10^{-5}$ with probability 0.95. We validated this bound by evaluating the sum via direct approach at all points, and 100% of the samples had relative error $\leq 10^{-5}$.

Bibliography

[1] A. Alvera-Azcárate, A. Barth, D. Sirjacobs, and J.M. Beckers. Enhancing temporal correlations in eof expansions for the reconstruction of missing data using dineof. *Ocean Science*, 5:475–485, October 2009.

[2] A Alvera-Azcrate, A. Barth, M. Rixen, and J.M. Beckers. Reconstruction of incomplete oceanographic data sets using empirical orthogonal functions: application to the adriatic sea surface temperature. *Ocean Modelling*, 9(4):325 – 346, 2005.

[3] R.K. Beatson, J.B. Cherrie, and C.T. Mouat. Fast fitting of radial basis functions: Methods based on preconditioned GMRES iteration. *Advances in Computational Mathematics*, 11:253–270, 1999.

[4] J.M. Beckers and M. Rixen. Eof calculations and data filling from incomplete oceanographic datasets. *Journal of Atmospheric and Oceanic Technology*, 20, 2003.

[5] F. Bimbot and et al. A tutorial on text-independent speaker verification. *EURASIP Journal on Applied Signal Processing*, 4:430–451, 2004.

[6] F. Bimbot, I. Magrin-Chagnolleau, and L. Mathan. Second-order statistical measures for text-independent speaker identification. *Speech Communication*, 17:51–54, 1995.

[7] C. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., 2006.

[8] C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[9] L. Burget, S. Cumani, O. Glembak, P. Matejka, and N. Brummer. Discriminatively trained probabilistic linear discriminant analysis for speaker verification. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[10] L. Burget, P. Matejka, P. Schwarz, O. Glembek, and J. Cernocky. Analysis of feature extraction and channel compensation in a GMM speaker recognition system. *IEEE Transactions on Audio, Speech, and Language Processing,*, 15(7):1979 –1986, Sep. 2007.

[11] D. Cai, X. He, and J. Han. Efficient kernel discriminant analysis via spectral regression. In *IEEE International Conference on Data Mining*, pages 427–432. IEEE Computer Society, 2007.

[12] J.P. Campbell. Speaker recognition: a tutorial. *Proceedings of the IEEE*, 85(9):1437–1462, Sep 1997.

[13] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-carrasquillo. Support vector machines for speaker and language recognition. *Computer Speech and Language*, 20:210–229, 2006.

[14] W.M. Campbell, J.P. Campbell, D.A. Reynolds, E. Singer, and P.A. Torres-Carrasquillo. Support vector machines for speaker and language recognition. *Computer Speech & Language*, 20:210 – 229, 2006. Odyssey 2004: The speaker and Language Recognition Workshop.

[15] W.M. Campbell, D.E. Sturim, D.A. Reynolds, and A. Solomonoff. SVM based speaker verification using a GMM supervector kernel and NAP variability compensation. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 1, 2006.

[16] S. Canu, S. Grandvalet, V. Guigue, and A. Rakotomamonjy. SVM and kernel methods Matlab toolbox. Perception Systmes et Information, France, 2005.

[17] B. Catanzaro, N. Sundaram, and K. Keutzer. Fast support vector machine training and classification on graphics processors. In *International Conference on Machine Learning*, pages 104–111, 2008.

[18] Il-Gyo Chong and Chi-Hyuck Jun. Performance of some variable selection methods when multicollinearity is present. *Chemometrics and Intelligent Laboratory Systems*, March 2005.

[19] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, 24(5):603–619, May 2002.

[20] L. Csató and M. Opper. Sparse on-line Gaussian processes. *Neural Comput.*, 14(3):641–668, 2002.

[21] N. de Freitas, Y. Wang, M. Mahdaviani, and D. Lang. Fast Krylov methods for n-body learning. In *Advances in Neural Information Processing Systems*, 2005.

[22] Defense. *The DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT)*. DARPA-ISTO, 1990.

[23] N. Dehak, R. Dehak, J. Glass, D. Reynolds, and P. Kenny. Cosine similarity scoring without score normalization techniques. In *Proc Odyssey Speaker and Language Recognition Workshop*, June 2010.

[24] N. Dehak, R. Dehak, P. Kenny, N. Brummer, P. Ouellet, and P. Dumouchel. Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. In *Interspeech*, pages 1559–1562, 2009.

[25] N. Dehak, P.J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788 –798, may 2011.

[26] Daniel P. W. Ellis. PLP and RASTA (and MFCC, and inversion) in Matlab, 2005. online web resource.

[27] D. Erdogmus, II Hild, K.E., and J.C. Principe. Independent components analysis using Rényi's mutual information and legendre density estimation. In *International Joint Conference on Neural Networks*, volume 4, pages 2762 –2767, 2001.

[28] R. Everson, P. Cornillonz, and A. Webbery. An empirical eigenfunction analysis of sea surface temperatures in the western north atlantic. *Journal of Physical Oceanography*, 1997.

[29] A.C. Faul, G. Goodsell, and M.J.D. Powell. A Krylov subspace algorithm for multiquadric interpolation in many dimensions. *IMA Journal of Numerical Analysis*, 25:1–24(24), 2005.

[30] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. In *Conference on Computer Vision and Pattern Recognition Workshop,*, page 178, 2004.

[31] L. Fei-Fei and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2005.

[32] D.P. Foster and R.A. Stine. Variable selection in data mining: Building a predictive model for bankruptcy. *Journal of the American Statistical Association*, pages 303–313, 2004.

[33] D. Garcia-Romero and C. Espy-Wilson. Joint factor analysis for speaker recognition reinterpreted as signal coding using overcomplete dictionaries. In *Proc Odyssey Speaker and Language Recognition Workshop*, June 2010.

[34] M. Gibbs and D. Mackay. Efficient implementation of Gaussian processes. Technical report, 1997.

[35] E. Gokcay and J.C. Principe. Information theoretic clustering. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(2):158–171, Feb 2002.

[36] N. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227(18):8290–8313, September 2008.

[37] N.A. Gumerov and R. Duraiswami. Fast radial basis function interpolation via preconditioned Krylov iteration. *SIAM Journal on Scientific Computing*, 29(5):1876–1899, 2007.

[38] T. Hansen. mgstat: A geostatistical matlab toolbox, 2004. online web resource.

[39] R. He, R.H. Weisberg, H. Zhang, F.E. Muller-Karger, and R.W. Helber. A cloud-free, satellite-derived, sea surface temperature analysis for the west florida shelf. *Geophysical Research Letters*, 30(15), August 2003.

[40] A. Hegde, T. Lan, and D. Erdogmus. Order statistics based estimator for Renyi entropy. *IEEE Workshop on Machine Learning for Signal Processing*, pages 335–339, Sept. 2005.

[41] A.O. Hero, B. Ma, O. Michel, and J. Gorman. Alpha divergence for classification, indexing and retrieval. Technical report, University of Michigan, 2001.

[42] E.H. Isaaks and R.M. Srivastava. *Applied Geostatistics*. Oxford University Press, 1989.

[43] R. Jenssen, II Hild, K.E., D. Erdogmus, J.C. Principe, and T. Eltoft. Clustering using renyi's entropy. *International Joint Conference on Neural Networks*, 1:523 – 528, 2003.

[44] A. Kaplan, Y. Kushnir, and M.A. Cane. Reduced space optimal interpolation of historical marine sea level pressure: 1854–1992. *Journal of Geophysical Research*, 1997.

[45] P. Kenny. Bayesian speaker verification with heavy tailed prior. In *Proc Odyssey Speaker and Language Recognition Workshop*, June 2010.

[46] P. Kenny, G. Boulianne, P. Ouellet, and P. Dumouchel. Speaker and session variability in gmm-based speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(4):1448 –1460, 2007.

[47] P. Kenny, P. Ouellet, N. Dehak, V. Gupta, and P. Dumouchel. A study of interspeaker variability in speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing,*, 16(5):980–988, July 2008.

[48] T. Kinnunen and H. Li. An overview of text-independent speaker recognition: From features to supervectors. *Speech Communication*, 52:12 – 40, 2010.

[49] D.G. Krige. A statistical approach to some basic mine valuation problems on the witwatersrand. *Journal of the Chemical, Metallurgical and Mining Society of South Africa*, 52(6):119–139, December 1951.

[50] N. Lawrence, M. Seeger, and R. Herbrich. Fast sparse Gaussian process methods: The informative vector machine. In *Advances in Neural Information Processing Systems*, pages 609–616, 2003.

[51] D. Lee, A. Gray, and A. Moore. Dual-tree fast Gauss transforms. In *Advances in Neural Information Processing Systems 18*, pages 747–754. 2006.

[52] S. Lophaven, H. Nielsen, and J. Sndergaard. Dace, a Matlab kriging toolbox, 2002. online web resource.

[53] D.G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–1157, 1999.

[54] S. Lukasik. Parallel computing of kernel density estimates with MPI. In *International conference on Computational Science*, pages 726–733. Springer-Verlag, 2007.

[55] Yong M., Yoshinori K., K. Kinoshita, S. Lao, and M. Kawade. Sparse Bayesian regression for head pose estimation. *International Conference on Pattern Recognition*, 3:507–510, 2006.

[56] J. Marron and M. Wand. Exact mean integrated squared error. *The Annals of Statistics*, 20(2):712–736, 1992.

[57] C.R. McClain, M.L. Cleave, G.C. Feldman, W.W. Greg, S.B. Hooker, and N. Kuring. Science quality seawifs data for global biosphere research. *Sea Technology*, 39:10 – 16, 1998.

[58] V. Morariu, B.V. Srinivasan, V.C. Raykar, R. Duraiswami, and L. Davis. Automatic online tuning for fast Gaussian summation. In *Advances in Neural Information Processing Systems*, 2008.

[59] R. Moyeed and A. Papritz. An empirical comparison of kriging methods for nonlinear spatial point prediction. *Mathematical Geology*, 34(4):365–386, May 2002.

[60] I. Murray. Gaussian processes and fast matrix-vector multiplies. In *Numerical Mathematics in Machine Learning workshop*, 2009.

[61] R. Murtugudde, S.R. Signorini, J.R. Christian, A.J. Busalacchi, C.R. McClain, and J. Picaut. Ocean color variability of the tropical indo-pacific basin observed by seawifs during 1997-1998. *Journal of Geophysical Research*, 104:18351–18366, 1999.

[62] Y. Notay. Flexible conjugate gradients. *SIAM J. Sci. Comput.*, 22(4):1444–1460, 2000.

[63] NVIDIA. *NVIDIA CUDA Programming Guide 3.2*. 2010.

[64] J. Ohmer, F. Maire, and R. Brown. Implementation of kernel methods on the GPU. In *Digital Image Computing: Technqiues and Applications,*, pages 543–550, Dec 2005.

[65] M.K. Omar and J. Pelecanos. A novel approach to detecting non-native speakers and their native language. In *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 4398 –4401, march 2010.

[66] G. Omer, R. Rosales, and B. Krishnapuram. Learning rankings via convex hull separation. In *Advances in Neural Information Processing Systems*, pages 395–402, 2006.

[67] J.C. Principe, J. Fisher, and D. Xu. *Information theoretic learning*. Wiley-Interscience, 2000.

[68] A. Ranganathan and M. Yang. Online sparse matrix Gaussian process regression and vision applications. In *European Conference on Computer Vision*, pages 468–482. Springer-Verlag, 2008.

[69] C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005.

[70] V. Raykar. *Scalable machine learning for massive datasets: Fast summation algorithms*. PhD thesis, Department of computer science, University of Maryland, College Park, 2007.

[71] V. Raykar, R. Duraiswami, and B. Krishnapuram. A fast algorithm for learning a ranking function from large-scale data sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1158–1170, 2008.

[72] V.C. Raykar and R. Duraiswami. The improved fast Gauss transform with applications to machine learning. In *Large Scale Kernel Machines*, pages 175–201, 2007.

[73] D.A. Reynolds, T. Quatieri, and R. Dunn. Speaker verification using adapted Gaussian mixture models. In *Digital Signal Processing*, 2000.

[74] M.R. Roman, W.C. Boicourt, D.G. Kimmel, W.D. Miller, J.E. Adolf, J. Bichy, L.W. Harding, Jr., E.D. Houde, S. Jung, and X. Zhang. Chesapeake bay plankton and fish abundance enhanced by hurricane isabel. *EOS Transactions*, 86:261–265, 2005.

[75] R. Rosipal and N. Kramer. Overview and recent advances in partial least squares. In *Subspace, Latent Structure and Feature Selection Techniques, Lecture Notes in Computer Science*, pages 34–51. Springer, 2006.

[76] R. Rosipal and L.J. Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. *J. Mach. Learn. Res.*, 2:97–123, March 2002.

[77] R. Rosipal and L.J. Trejo. Kernel PLS-SVC for linear and nonlinear classification. In *20th International Conference on Machine Learning*, pages 640–647, 2003.

[78] Y. Saad. *Numerical Methods for Large Eigenvalue Problems*. Manchester University Press, 1992.

[79] Y. Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, 1993.

[80] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, 2003.

[81] W.R. Schwartz, A. Kembhavi, D. Harwood, and L.S. Davis. Human detection using partial least squares analysis. In *International Conference on Computer Vision*, 2009.

[82] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization (Wiley Series in Probability and Statistics)*. Wiley-Interscience, September 1992.

[83] M Seeger, C.K.I Williams, N Lawrence, and S. Dp. Fast forward selection to speed up sparse gaussian process regression. In *Workshop on AI and Statistics 9*, 2003.

[84] M. Senoussaoui, P. Kenny, N. Dehak, and P Dumouchel. An i-vector extractor suitable for speaker recognition with both microphone and telephone speech. In *Proc Odyssey Speaker and Language Recognition Workshop*, June 2010.

[85] S. Sheather and M. Jones. A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society*, 53:683–690, 1991.

[86] S.R. Signorini, R. Murtugudde, C.R. McClain, J.R. Christian, J. Picaut, and A.J. Busalacchi. Biological and physical signatures in the tropical and subtropical atlantic. *Journal of Geophysical Research*, 104:18367–18382, 1999.

[87] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, April 1986.

[88] T. Sim, S. Baker, and M. Bsat. The cmu pose, illumination, and expression database. *IEEE transactions on Pattern Analysis and Machine Inference*, 25(12):1615–1618, Dec. 2003.

[89] V. Simoncini and D.B. Szyld. Flexible inner-outer Krylov subspace methods. *SIAM J. Numer. Anal.*, 40(6):2219–2239, 2002.

[90] J. Sivic, B.C. Russell, A.A. Efros, A. Zisserman, and W.T. Freeman. Discovering object categories in image collections. In *IEEE International Conference on Computer Vision*, 2005.

[91] E. Snelson and Z. Ghahramani. Sparse Gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264, 2006.

[92] E. Snelson and Z. Ghahramani. Local and global sparse gaussian process approximations. In *Artificial Intelligence and Statistics (AISTATS)*, 2007.

[93] F. Soong, A. Rosenberg, L. Rabiner, and B. Juang. A vector quantization approach to speaker recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 10, pages 387–390, Apr 1985.

[94] B.V Srinivasan and R. Duraiswami. Efficient subset selection via the kernelized Rényi distance. In *IEEE International Conference on Computer Vision*, pages 1081–1088, September 2009.

[95] B.V. Srinivasan and R. Duraiswami. Scaling kernel machine learning algorithm via the use of GPUs. In *GPU Technology Conference*. NVIDIA Research Summit, 2009.

[96] B.V Srinivasan and R. Duraiswami. Kernelized rényi distance for similarity scoring and speaker recognition. Technical report, University of Maryland - CS-TR-4994, 2011.

[97] B.V. Srinivasan, R. Duraiswami, and N. Gumerov. Fast matrix-vector product based fgmres for kernel machines. In *Copper Mountain Conference on Iterative Methods*, 2010.

[98] B.V. Srinivasan, R. Duraiswami, and R. Murtugudde. Efficient kriging for real time spatio-temporal kriging. In *Conference on Probability and Statistics in Atmospheric sciences.*, 2010.

[99] B.V. Srinivasan, R. Duraiswami, and D.N. Zotkin. Kernelized Rényi distance for speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2010.

[100] B.V. Srinivasan, D Garcia-Romero, D.N. Zotkin, and R. Duraiswami. Kernel partial least squares framework for speaker recognition. In *12th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, 2011.

[101] B.V. Srinivasan, Q. Hu, and R. Duraiswami. GPUML: Graphical processors for speeding up kernel machines. In *Workshop on High Performance Analytics - Algorithms, Implementations, and Applications*. Siam International Conference on Data Mining, 2010.

[102] B.V Srinivasan, W.R. Schwartz, R. Duraiswami, and L.S. Davis. Partial least squares on graphical processor for efficient pattern recognition. Technical report, University of Maryland - CS-TR-4968, 2010.

[103] B.V. Srinivasan, D.N. Zotkin, and R. Duraiswami. A partial least squares based speaker recognition system. In *Snowbird Learning Workshop*, 2011.

[104] B.V. Srinivasan, D.N. Zotkin, and R. Duraiswami. A partial least squares framework for speaker recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[105] D. Steinkraus, I. Buck, and P. Simard. Using GPUs for machine learning algorithms. In *International Conference on Document Analysis and Recognition,*, volume 2, pages 1115–1120, Sep 2005.

[106] R. Teofilo, J. Martins, and M. Ferreira. Sorting variables by using informative vectors as a strategy for feature selection in multivariate regression. In *Journal of Chemometrics*, volume 23, pages 32–48.

[107] D. Thanh-Nghi and V. Nguyen. A novel speed-up SVM algorithm for massive classification tasks. In *IEEE International Conference on Research, Innovation and Vision for the Future*, pages 215–220, July 2008.

[108] A. Thayananthan, R. Navaratnam, B. Stenger, P. Torr, and R. Cipolla. Multivariate relevance vector machines for tracking. In *European Conference on Computer Vision*, pages 124–138. Springer-Verlag, 2006.

[109] M. Tipping. The relevance vector machine. In *Advances in Neural Information Processing Systems*. Morgan Kaufmann, 2000.

[110] L. Torgo. Available at `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`.

[111] V. Vapnik. *The Nature of Statistical Learning Theory (Information Science and Statistics)*. Springer, 2nd edition, November 1999.

[112] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008.

[113] V. Volkov and J. Demmel. LU, QR and Cholesky factorizations using vector capabilities of gpus. Technical Report UCB/EECS-2008-49, EECS Department, University of California, Berkeley, May 2008.

[114] C. Williams and C. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, 1996.

[115] S. Wold, E. Johansson, and M. Cocch. PLS-partial least squares projections to latent structures. *3D QSAR in Drug Design*, pages 523 – 548, 1993.

[116] L . Wolf, T. Hassner, and Y. Taigman. Descriptor Based Methods in the Wild. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, Marseille, France, 2008.

[117] L. Wolf, T. Hassner, and Y. Taigman. The one-shot similarity kernel. In *IEEE 12th International Conference on Computer Vision*, pages 897 –902, 2009.

[118] D. Xu, J.C. Principe, J. Fisher, and H. Wu. A novel measure for independent component analysis (ICA). In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1161–1164, May 1998.

[119] C. Yang, C. Duraiswami, and L. Davis. Efficient kernel machines using the improved fast gauss transform. In *Advances in Neural Information Processing Systems*, 2004.

[120] C. Yang, R. Duraiswami, and L. Davis. Efficient mean-shift tracking via a new similarity measure. In *IEEE International Conference on Computer Vision and Pattern Recognition*, volume 1, pages 176–183, June 2005.