# Combining Kernels for Classification

**Darrin P. Lewis**

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

**COLUMBIA UNIVERSITY**

2006

# ABSTRACT

## Combining Kernels for Classification

**Darrin P. Lewis**

Drawing inferences from large, heterogeneous data sets requires a theoretical framework that is capable of representing, for example, DNA and protein sequences, protein structures, microarray expression data, various types of interaction networks, etc. Recently, a class of algorithms known as *kernel methods* has emerged as a powerful framework for combining diverse types of data.

The power and current popularity of kernel methods stem in part from their ability to handle diverse forms of structured inputs, including vectors, graphs, and strings. The support vector machine (SVM) algorithm is the most popular kernel method, due to its theoretical underpinnings and strong empirical performance on a wide variety of classification tasks. Recently, several methods have been proposed for combining kernels from heterogeneous data sources. Specifically, several recently described extensions allow the SVM to assign relative weights to various data sets, depending upon their utilities in performing a given classification task. However, all of these methods produce stationary combinations; i.e., the relative weights of the various kernels do not vary among input examples.

In this work, we describe, implement and validate a method for combining multiple kernels in a nonstationary fashion, where the kernel function combination varies depending on the input. The approach uses a large-margin latent-variable generative probabilistic model within the maximum entropy discrimination (MED) framework. In this method, parameter estimation is rendered tractable by variational bounding and an iterative optimization procedure. Here, we propose an MED Hilbert space

Gaussian mixture model, in which each component is implicitly mapped via a Mercer kernel function, and we show that the support vector machine is a special case of this model. The mixture model allows us to combine a given set of kernels in a nonlinear and nonstationary manner, while avoiding overfitting by regularization. We also derive an efficient sequential minimal optimization algorithm for discriminative parameter estimation.

We empirically investigate the performance of the SVM and its variants on numerous multi-kernel learning tasks, ranging from illustrative synthetic data sets, to commonly used benchmark data sets, to the real-world computational biology problem of protein function annotation. In the majority of cases, and without any particular tuning of the algorithm, our new technique outperforms existing methods.

# Contents

# List of Figures

# List of Tables

# ACKNOWLEDGEMENTS

I would like to thank my advisor William Stafford Noble, who stood by me through his transition to the University of Washington and my advisor Tony Jebara, who took me into his group as a full fledged member, and to both for their direction, support, and guidance during my doctoral education. The complementary experience to which they have exposed me has been invaluable.

I would also like to thank the members of both labs, from whom I have gotten advice, friendship, and an open exchange of ideas during my relatively solitary research.

Finally, I would like to thank my family, especially my wife, who have supported me in the rather unusual lifestyle of a student and encouraged me to continue through the most difficult times. I look forward to seeing them again.

The text for Chapter 2 is a reprint of the material as it was submitted to *Bioinformatics* [Lewis, Jebara, and Noble, 2006c]. The thesis author is primary author for the work and the co-authors directed and supervised the research for this chapter.

The text for Chapter 3 is a reprint of the material as it was submitted to the *Journal of Machine Learning Research* [Lewis, Jebara, and Noble, 2006a]. The thesis author is primary author for the work and the co-authors directed and supervised the research for this chapter.

The text for Chapter 4 is largely a reprint of the material as it was accepted to the *International Conference on Machine Learning* [Lewis, Jebara, and Noble, 2006b]. The thesis author is primary author for the work and the co-authors directed and supervised the research for this chapter.

To my wife, Susanne.

# Chapter 1

# Introduction

## 1.1 Motivation

Amongst practitioners, there is a vital need to learn from heterogeneous data sets. This need is fueled by the increasing amount of data being generated by different processes, that potentially inform different aspects of a learning problem. As an example, consider the computational biology problem of protein function annotation. Numerous wet lab experiments have provided a wide variety of data pertaining to the same finite set of proteins. Each type of measurement, e.g., DNA sequence, three dimensional structure, subcellular location, protein domains, interaction networks, etc., offers a different set of discriminative features for classification. The practitioner should have the freedom to use all of these data to inform a classifier and should expect the learning algorithm to use the data in a sensible way.

Kernel methods [Schölkopf et al., 1999], by providing a sort of canonical form for data sets, provide a large step toward the goal of learning from heterogeneous data. In kernel methods, the kernel function is the sole interface between the learning algorithm and the data. The kernel isolates the learning algorithm from the details of

the data representation and from the distance metric. Kernels have allowed machine learning algorithms to be modularized, but they have other important advantages such as providing a compact, implicit mapping to high- and even infinite-dimensional feature spaces and permitting linear classifiers to operate in these non-linear spaces.

In this thesis, we consider the combination of kernel functions for the task of classification. Previous applications in this area [Pavlidis et al., 2001, 2002, Joachims et al., 2001, Lanckriet et al., 2004a,b] have shown that combining kernels can—and often does—improve classifier performance. We propose a new, more powerful technique for combining kernels and evaluate its performance in several empirical studies [Lewis et al., 2006b,a].

As pictorial motivation, we present the basic scenario in the following illustration (Figures 1.1-1.3). Consider a data set that consists of height and weight measurements taken at two laboratories. The measurements are taken from a sample of male and female subjects at birth and maturity. Though this is a fabricated example, we expect four roughly Gaussian clusters corresponding to female birth, male birth, female adult, male adult. In our example, however, an "unintended" variable has been introduced. One of the laboratories has a miscalibrated scale that was used for all measurements.

Now let us assume that we wish to learn to classify male and female examples given only height and weight measurements. We choose to classify with a log ratio of likelihoods from two Gaussian mixture models (one for male, one for female). Using prior knowledge, we choose two Gaussians per model, for birth and adult. We do not know about the miscalibrated scale.

Figure 1.1 shows the 2-dimensional data. Figure 1.2 illustrates the result we can expect if we train the Gaussian mixture models independently, using maximum likelihood parameter estimation. The log ratio classifier would be confounded by the irrelevant weight scale miscalibration and would yield poor discrimination. This occurs because the measurement is so far off as to make the adult-child grouping more likely under the Gaussians, than the gender groupings. Maximum likelihood estima-


Figure 1.1: **Example of Gaussian data with an unintended variable.** This figure shows an illustrative example that emphasizes the need for discriminative parameter estimation for classification with mixtures. The data contains an "unintended" variable.

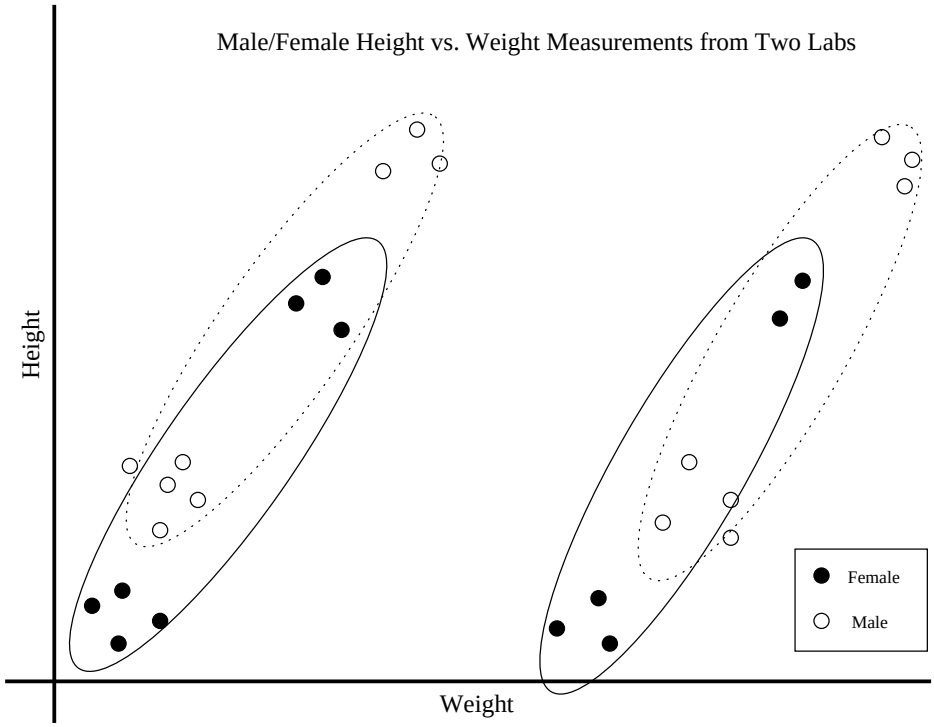


Figure 1.2: **Example of an ML Gaussian mixture with an unintended variable.** This figure continues the illustrative example that emphasizes the need for discriminative parameter estimation for classification with mixtures. We see two two-component Gaussian mixture models trained independently with maximum likelihood. Classification with these models will be poor.

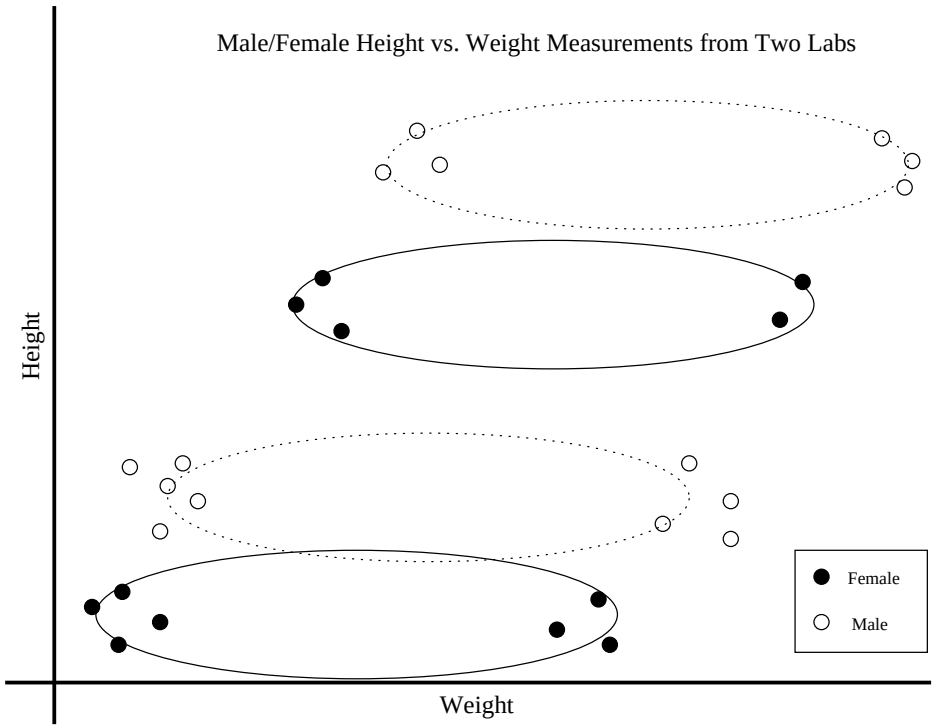


Figure 1.3: **Example of an MED Gaussian mixture with an unintended variable.** This figure continues the illustrative example that emphasizes the need for discriminative parameter estimation for classification with mixtures. We see two two-component Gaussian mixture models trained discriminatively with MED. Classification with these models will be correct.

tion of the male and female models ignores the fact that the classes are not separated properly. In contrast, the latent MED Gaussian mixture we propose would be trained discriminatively, using maximum entropy parameter estimation. The classifier would learn to ignore the weight variable, because given the model, it is not as discriminative as the height. The learned Gaussians would appear as in Figure 1.3 and classification would thus be successful. In fact, this illustrative example forms the basis for a synthetic data experiment in Chapter 4.

This is an artificial example. However, it highlights a real issue in machine learning. It is almost always the case that a probabilistic model is a simplification of the true distribution being modeled. When model mismatch arises, at least in a discriminative learning scenario, it is clear that optimizing the parameter estimation for the discriminative objective rather than the generative one is a viable option. In the extreme case, the support vector machine (SVM) [Vapnik, 1998] abandons the goal of modeling the input distribution altogether, and only maximizes a measure of discriminative performance. This has proven a simple and successful strategy. However, we believe that modeling can be judiciously reintroduced to exploit structure in a data set. To address this problem, we offer latent MED. We further propose using kernelized latent MED to permit a Gaussian mixture model in which each Gaussian resides in a different feature space. This results in a nonstationary kernel combination, with novel benefits as we shall describe.

## 1.2 Contribution

The principal contribution of this thesis is nonstationary kernel combination (NSKC). NSKC gives the ability to exploit prior knowledge through the *a priori* choice of several kernels and a structured generative model. The model allows us to combine kernels in a more flexible manner than simple linear combination. The proposed *nonstationary* combination permits the kernel weight to vary over the input space, as

a posterior distribution over a mixture model. Each kernel has greatest influence over the classifier where it is most relevant to the input distribution. This is a generalization over existing techniques, as will be shown.

To achieve NSKC, we combine large margin discrimination with generative modeling in a probabilistic framework and apply the resulting classifiers to multi-kernel learning problems. We use the formalism of maximum entropy discrimination [Jaakkola, Meila, and Jebara, 1999b] to generalize upon support vector machines [Boser et al., 1992, Cortes and Vapnik, 1995, Vapnik, 1998].

The current research improves the state-of-the-art in several important ways. Though multi-kernel learning, large margin classification, and indeed maximum entropy discrimination are existing techniques, through this research, we have extended and applied them in an interesting new way. The prior work is discussed in Section 1.3 and our contributions are detailed in the following chapters. Here, we briefly highlight some of the key contributions of the research to set the stage for the later sections.

In Jaakkola, Meila, and Jebara [1999b] the ideas of maximum entropy discrimination (MED) were explicated in detail. That paper proposes the extension of MED for use with latent variable models, though specifics are omitted. Jebara [2004a] further explores latent MED and provides more detail. In this thesis, we propose a new latent MED variational bounding technique that provides a true bounded solution and a clearer understanding of the latent MED parameter estimation as a minimax problem. Our approach is principled and our derivations are presented in full. The new bounding technique provides a guarantee of convergence to a locally optimal solution.

Jebara [2004a] also offers the suggestion of a more efficient optimization technique for a latent MED mixture of Gaussians. The intent was to provide a starting point for further work, which was indeed required. We completely re-derived the update equations for a sequential minimal optimization (SMO) [Platt, 1999] algorithm for the

latent MED mixture of Gaussians using a more elegant formulation of the problem. We also discover and explicate several cases that were omitted from Jebara [2004a], so we cover the entire feasible space. We discovered that a standard SMO algorithm can fail to reach an optima in reasonable time, due to non-smoothness of the latent MED objective, and we provide a solution for that problem. As a result of these advances, we are able to provide the first implementation of SMO for latent MED problems. The software is written in C++ and is invoked from our MATLAB software using the MEX interface.

We introduced the technique of nonstationary kernel combination, using a latent MED mixture of Gaussians. Kernel combination techniques have, thus far, been linear combinations of kernels [Pavlidis et al., 2001, 2002, Lanckriet et al., 2002, Tsuda et al., 2005, Ong et al., 2005] or their eigenvectors [Cristianini et al., 2002, Ong et al., 2005]. The combination weights have been independent of the input to be classified, and thus are *stationary*. We introduce dependence of the combination weights on the input space, creating a *nonstationary* combination of kernels [Lewis, Jebara, and Noble, 2006b]. This new technique is interesting because it is capable of solving problems that existing techniques cannot solve, as illustrated in synthetic data experiments. Thus far, empirical results with NSKC on common benchmarks and real problems have usually been better than existing techniques, without the need for particular tuning of the algorithm.

We conducted several empirical studies using kernel combination, including one set of experiments [Lewis, Jebara, and Noble, 2006c] that illustrates that learning kernel combination weights via optimization is not always superior to using a simple unweighted combination. This study is particularly relevant to practitioners due to the time required to to compute the combination weights for large data sets and due to the current lack of freely available software to perform the optimization. In our experiments, the unweighted combination was often as good as an optimized combination. We discuss some cases in which we observed an advantage for the

optimization. This research also examined the effect of missing data with kernel combination. We found that the simple unweighted combination was quite robust to missing data. This study may encourage experimentalists to try simple kernel combination techniques when optimization is either inefficient or unavailable.

We have created a complete MATLAB/C++ software implementation that we will make publicly available for academic use.

## 1.3 Prior Work

We build upon a great wealth of recent work in the areas of kernel methods [Schölkopf et al., 1999, Schölkopf and Smola, 2002], large margin classification [Vapnik, 1998], probabilistic modeling and inference [Pearl, 1998, Jordan et al., 1999], and optimization [Boyd and Vandenberghe, 2003, Platt, 1999]. There has been explosive progress, of late, in all of the above areas. This makes machine learning and its application an exciting field, but also makes it difficult to stay ahead of the wave of theoretical and empirical results and the flood of data. The field has responded in several ways. One is to develop faster and more accessible software to address the needs of practitioners to evaluate ever-growing data repositories [Platt, 1999, Joachims, 1998]. Another is to increase the complexity of our techniques to learn more from existing data [Jaakkola and Haussler, 1998, Jaakkola et al., 1999a,c, Taskar et al., 2003, Altun et al., 2003, Tsochantaridis et al., 2004, Jebara, 2004a]. Some approaches facilitate both of these goals, to make tractable approximate algorithms that solve complex problems [Dempster et al., 1977, Jordan et al., 1999, Jebara, 2004a].

Much of the current enthusiasm stems from a great recent triumph in machine learning, the support vector machine (SVM) [Boser et al., 1992, Cortes and Vapnik, 1995, Vapnik, 1998, Cristianini and Shawe-Taylor, 2000]. To understand the significance of the SVM, we should take a brief look at the field prior to its discovery. Artificial intelligence (AI), of which machine learning is an offspring, had suffered a

varied past, due to cycles of irrational exuberance followed by overcorrection.

The field of AI had promised great results, which never seemed to materialize. The late 1950's saw the advent of the perceptron algorithm [Rosenblatt, 1959], which was a simplified model of the neuron. The perceptron was seen as having great potential until Minsky and Papert [1967, 1969] showed its theoretical limitations. Decades later, the research community responded with (artificial) neural networks (ANNs), essentially perceptrons cascaded into multilayer networks. Progress was slow until feed-forward networks and the backpropagation training algorithm [Werbos, 1974] were discovered. These techniques were popularized in the early 1980's [Rumelhart et al., 1986]. ANNs still suffered several problems, even with backpropagation. The primary problems were data representation, choice of network topology, non-convex (non-global) optimization, and overfitting. Though neural networks are still in use, the solutions to their problems came largely through another class of algorithms.

The support vector machine (SVM) [Vapnik, 1998] has had enormous success with classification problems drawn from many fields, such as biology [see Noble, 2004], text mining [Dumais, 1998], image recognition [Osuna et al., 1997], and others. This success can be attributed to the empirical performance of the algorithm, the accessibility of quality software [Pavlidis et al., 2004, Joachims, 1998], the flexibility imparted by kernel methods [Schölkopf et al., 1999], and the sound theoretical underpinnings of the technique [Vapnik, 1998]. Much of the appeal of the SVM comes from its relatively simple convex optimization [Boyd and Vandenberghe, 2003], which balances the empirical loss of the classifier with its expected generalization.

Rallied by the popularity of the SVM, researchers have sought to extend its domain to include a greater variety of problems. These extensions include the ability to perform regression [Drucker et al., 1997, Schölkopf et al., 1999], multi-class classification [Weston and Watkins, 1998], feature selection [Weston et al., 2001], and transduction [Gammerman et al., 1998, Joachims, 1999, Collobert et al., 2005], within the SVM optimization. Others have extended SVMs to structured data directly [Taskar et al.,

2003, Altun et al., 2003, Tsochantaridis et al., 2004] and through kernel functions [Jaakkola and Haussler, 1998, Leslie et al., 2002, 2003]. Still others extend SVMs to much larger problems by making scalable algorithms and implementations [Platt, 1999, Joachims, 1998], both in terms of time and storage. The availability of fast, free SVM implementations has accelerated its acceptance. The SVM has become so mainstream that classification services are offered for free on the web [Pavlidis et al., 2004].

Another active area of machine learning research has been generative probabilistic modeling and graphical models [Pearl, 1998, Jordan and Bishop]. Latent variable models are typically handled by expectation maximization [Dempster et al., 1977] or variational methods [Jordan et al., 1999] (of which EM is an example).

Recently, there have been various attempts to merge discriminative and generative techniques [Jaakkola and Haussler, 1998, Jaakkola et al., 1999a, 2000, Jebara and Pentland, 1998, Jebara, 2004a, Taskar et al., 2003, Altun et al., 2003]. This trend reflects the desire to exploit prior knowledge of the input distribution, and the tendency toward a unified generative-discriminative learning paradigm.

Due to the proliferation of numerous data sets pertaining to the same sets of entities, e.g., proteins, kernel combination techniques [Pavlidis et al., 2001, 2002, Cristianini et al., 2002, Lanckriet et al., 2002, Ong et al., 2005, Tsuda et al., 2005, Sonnenburg et al., 2006a] are becoming popular. Due to their abstraction from learning algorithms and their mathematical properties, combining kernels is a reasonable approach to learning from heterogeneous data. We should make a distinction between combining kernels and problem of "learning" the kernel. What many refer to as learning the kernel is actually learning some combination of kernels given *a priori*. Often, even the kernel combination techniques work with kernel matrices (Gram matrices), so they combine kernel evaluations, rather than kernel functions. However, some combination function is usually learned, so the combined kernel can be evaluated for test examples taken from the same individual kernel functions, given their

values. The SDP approach [Lanckriet et al., 2002] is transductive, so it can benefit from knowing the test examples during training.

## 1.4 Organization of Thesis

The remainder of the thesis is organized into three chapters drawn from our published and submitted work on combining kernels and a fourth chapter that concludes the thesis and discusses future work.

Chapter 2 is an empirical study of learning with heterogeneous data. The paper deals with conic combinations of kernels containing data from yeast protein sequence and structure. We call this stationary kernel combination because the combination weights do not vary across the input space. Existing techniques available to biology researchers fall into this category. This study is important because practitioners may not have access to the more complicated SDP software and would not be in a position to perform a comparative evaluation of the techniques. The lack of freely available software and the time complexity of optimization may be slowing the adoption of multi-kernel learning techniques, but our results indicate that need not be the case. In this set of experiments, in all but the most perverse cases, the simplest technique performs very well. This result should spark some interest in the applied community. We also show some conditions under which the more complicated SDP technique shows its strength. In addition, we present a result that should encourage practitioners to use additional partial information when it is of good quality. Such is the case regarding protein structure data, which is in relatively short supply. The SVM was able to make use of an additional kernel matrix with as much as half of the data missing.

Chapter 3 describes maximum entropy discrimination in detail and derives our nonstationary kernel combination. We also present derivations for a sequential minimal optimization procedure for our latent MED Gaussian mixture, and share expe-

riences regarding the implementation.

Chapter 4 presents our novel nonstationary kernel combination technique in a comparative study with state-of-the-art techniques. The empirical results are taken over illustrative synthetic data sets, common benchmark data sets, and a real-world bioinformatics data set. This research is the first published work on nonstationary kernel combination and opens the way for further research in the direction of "adaptive" or input-dependent kernel combinations. Under roughly three fourths of the experimental conditions, the nonstationary combination outperformed existing techniques. The chapter contains the results accepted for publication and an additional as yet unpublished result.

Chapter 5 concludes the thesis by summarizing our contribution and reviewing our results. We also present some possible future directions for the work.

# Chapter 2

# Stationary Kernel Combination

In this chapter, we present a comparison of some existing techniques for combining kernels. These techniques use a conic combination of kernel matrices. They do not employ nonstationarity, i.e., the combination weights do not vary over the input space. Nonetheless, the SDP technique [Lanckriet et al., 2002] is the current state-of-the-art. The primary drawbacks of using SDP are the time required for the optimization and the lack of a freely available implementation. For practitioners in computational biology, in which large data sets and multiple kernels are common, these drawbacks of SDP cannot be ignored.

The following empirical study was submitted to the journal *Bioinformatics* [Lewis, Jebara, and Noble, 2006c]. We believe that this pragmatic comparison of SDP to the extremely simple unweighted combination will be of value to the applied community.

## 2.1 Introduction

It is by now a truism to point out that biological data is being produced at a rapid rate. Less obvious, but equally daunting, is the large variety of types of biological data being produced. Traditional statistical methods that assume Gaussian distributions, or engineering methods that assume vector or matrix input do not obviously

generalize to data sets comprised of variable-length strings, vectors of real numbers, trees and networks. Consequently, much recent work has focused on the development of statistical and computational methods that are capable of drawing inferences from large, heterogeneous biological data sets.

Kernel methods [Schölkopf et al., 1999] provide a principled means to represent and hence draw inferences from diverse types of data. A kernel method represents a collection of arbitrarily complex data objects by using a so-called kernel function that defines the similarity between any given pair of objects. In practice, this means that a collection of $N$ objects can be sufficiently represented via an $N$-by-$N$ matrix of pairwise kernel values. This kernel matrix, hence, provides a sort of normal form: as long as a valid kernel function can be defined on a given data type, then any such data set can be represented as a kernel matrix. Kernel methods are algorithms that operate on kernel matrices, rather than on the raw data objects themselves.

By far the best-known kernel method is the support vector machine (SVM) algorithm [Boser et al., 1992, Vapnik, 1998, Cristianini and Shawe-Taylor, 2000]. The SVM is a supervised classification algorithm that learns by example to discriminate among two or more given classes of data. Within computational biology, SVMs have been applied to an increasing variety of problems, including remote protein homology detection, various types of gene expression analyses, splice site and alternative splicing detection, tandem mass spectrometry analysis, etc. [Noble, 2004].

In order to apply an SVM to a heterogeneous data set, kernels must be defined for each data type, and the kernel matrices must be combined algebraically. For example, Pavlidis *et al.* used this approach to combine microarray gene expression data and phylogenetic profiles in an unweighted fashion, applying mathematical operators to focus the SVM on within-data-set correlations among features while ignoring correlations between data sets [Pavlidis et al., 2001, 2002]. An unweighted sum of kernels has also been used successfully in the prediction of protein-protein interactions [Ben-Hur and Noble, 2005].

Recently, several research groups have proposed *multiple kernel learning* (MKL) methods that combine kernels within the SVM algorithm itself [Lanckriet et al., 2002, Bach et al., 2004, Ong et al., 2005, Sonnenburg et al., 2006a, Jebara, 2004b]. These methods formulate a single optimization procedure that simultaneously finds the SVM classification solution as well as weights on the individual data types in the heterogeneous set. Lanckriet et al. [2002] formulate the problem using semidefinite programming, whereas [Ong et al., 2005] formulate the problem using semi-infinite linear programming. These approaches have been successful in various bioinformatics tasks, including yeast protein functional classification [Lanckriet et al., 2004b,a], protein structure classification [Borgwardt et al., 2005], protein subcellular localization [Zien and Ong, 2006] and alternative splicing recognition [Sonnenburg et al., 2006b].

In general, MKL methods that assign weights to individual data types have some practical disadvantages. The SDP approach of Lanckriet *et al.* requires large amounts of memory. This problem was essentially solved by Bach et al. [2004], but the resulting algorithm is still quite slow. Other MKL methods are also slow. Furthermore, all of these algorithms are more complicated to program than a simple SVM.

The current work aims to answer the question: are MKL methods worth the additional effort, relative to using an unweighted sum of kernels? Furthermore, if MKL methods are useful, then we would like to be able to characterize the situations in which they should be used.

To answer this question, we focus on a particular task: predicting Gene Ontology (GO) terms using a combination of amino acid sequence and protein structural information. This task has intrinsic interest: considering the amount of effort currently being expended on inferring protein structures, it is interesting to quantify the extent to which protein structure improves upon our ability to draw inferences about proteins, relative to inferences drawn from sequence alone. Furthermore, the problem has two characteristics that are useful in the context of this study. First, by restricting ourselves to two kernels, it is possible to explore systematically the space

of possible linear combinations. Second, since we know that structure is generally more informative than sequence, we can expect reasonably consistent behavior of our kernel combinations across a wide variety of classification tasks.

Our experiments show, first, that protein structure is more informative, i.e., more predictive of function, than protein sequence. This is an expected, but comforting result. The structure kernel that we employ uses MAMMOTH [Ortiz et al., 2002], which is a structural alignment algorithm that considers only the protein backbone, ignoring the side chains that differ among amino acids. Hence, this kernel, by its design, is fairly independent of the sequence kernel. Nonetheless, even without side chain information, the structure kernel provides better recognition performance than the sequence kernel on all 56 GO terms that we consider. These terms come from all three GO hierarchies—molecular function, biological process and cellular compartment.

Perhaps more surprisingly, we find that, for this two-kernel task, the unweighted average of kernels performs slightly *better* than a more sophisticated method that assigns weights to the kernels. Indeed, by systematically considering various relative weights, we are able to demonstrate that, for this particular task, no kernel-weighting scheme can perform much better than the simple unweighted sum of kernels.

On the other hand, in a follow-up experiment, we demonstrate that MKL is indeed helpful in some circumstances. Specifically, we consider the case in which additional, noisy kernels are added to the sequence and structure kernels. As we add more noise to the system, the performance of the unweighted average deteriorates. In contrast, the weighted kernel approach learns to down-weight the noise kernels, and hence continues to work well.

Finally, in a separate experiment, we investigate the performance of both kernel combination methods in the presence of missing data. In practice, we have sequence information for many more proteins than we have structure information. Such missing data is common in genome-wide data sets, and the probability that any one gene

or protein will have missing data increases as we include more data types in a single classification experiment. Hence, it is interesting to ask how well an SVM performs when one of the kernels in the combination contains missing data. In general, however, SVMs do not provide a mechanism for handling missing data. We consider three simple techniques for representing missing examples in a kernel matrix. Our experiments do not definitely show that one of these three techniques is best; however, we do demonstrate that, using any of the three proposed methods for handling missing data and using either a weighted or unweighted kernel combination, the SVM performance degrades fairly gradually as the percentage of missing data in the structure kernel increases.

This empirical study aims to provide guidance to users of SVM classifiers, as well as to suggest avenues for further research. Perhaps our most important practical conclusion is that the simple, unweighted sum of kernels can provide remarkably robust classification performance. Only when we used a relatively large collection of data types, with some data that were less relevant to the task at hand, did a weighted kernel combination method add value. The primary benefit of the optimizing the weights was to eliminate a kernel completely. This suggests that some experimentation with individual kernels may reduce reliance on optimization software. Once a set of relevant kernels is selected, unweighted combination may be sufficient. Given the low computational cost of this approach, it seems warranted. Our results also suggest caution in interpreting the specific weights assigned to each data type by a weighted kernel approach, since the SVM performance does not vary dramatically as the kernel weights change.

## 2.2 Algorithms

We provide here a brief, non-technical overview of the SVM, followed by descriptions of the two methods for handling heterogeneous data. More detail on SVMs and kernel

methods can be found in, e.g., Cristianini and Shawe-Taylor [2000], Schölkopf et al. [1999].

### 2.2.1 Support vector machines

Applying an SVM to a classification problem consists of two phases: training and prediction. During training, the SVM takes as input a data set in which each example is a fixed-length vector. Furthermore, each example must have an associated binary label. We use "+1" to denote the positive class and "-1" to denote the negative class. If each vector contains $m$ values, then we say that the data resides in an $m$-dimensional space called the *input space*.

The SVM training algorithm searches for a plane (or, when $m > 3$ a hyperplane) in the input space that separates the positive from the negative examples. Learning theory suggests that, when many such hyperplanes exist, an optimal procedure selects the hyperplane that is farthest from any training example. This particular hyperplane is known as the *maximum margin hyperplane.* The problem of selecting, for a given data set, the maximum margin hyperplane can be formulated and solved efficiently using quadratic programming. This optimization constitutes the SVM training phase.

Having identified this separating hyperplane, the prediction phase takes as input a second data set of length-$m$ vectors. The goal of this phase is to predict the associated +1/-1 label for each of the test examples. Prediction is accomplished by simply asking on which side of the separating hyperplane each test example falls.

This description of the SVM algorithm leaves out many details, but should be sufficient for our subsequent discussion. Most notably, we have not described how the SVM works when no separating hyperplane exists. Briefly, this situation is handled in two ways. The first solution involves introducing a so-called "soft margin," which allows a subset of the training data to fall on the "wrong" side of the hyperplane; e.g., a few examples labeled "+1" might lie on the "-1" side of the hyperplane, and vice versa. The second solution involves introducing a kernel function, which we describe

next.

### 2.2.2 Kernel methods

Generically, a kernel function is simply a function that defines the similarity between a given pair of objects. Denoted $K(x, y)$, a large value indicates that $x$ and $y$ are similar, and a small value indicates that they are dissimilar. In the context of kernel methods in machine learning, the kernel function must have certain mathematical properties; namely, for all possible data sets, the matrix of all-vs-all kernel values must have non-negative, real eigenvalues, which ensures that the kernel is a valid metric.

The fundamental idea of a kernel method is simple but somewhat subtle. Say that you have a collection of $n$ vectors, each of length $m$. This data can be written as an $m$-by-$n$ matrix. Given a kernel function $K(\cdot, \cdot)$, we can compute the similarity between all pairs of vectors in the data set. These kernel values can then be written as an $n$-by-$n$ matrix, called the kernel matrix. For an algorithm to be a kernel method, it must be possible to show that the kernel matrix is a sufficient representation of the data. In other words, if an algorithm is a kernel method, then it should be possible to discard the original data matrix and still run the algorithm, using only the kernel matrix.

The canonical kernel function is the scalar product (a.k.a. the dot product or vector product) $K(x, y) = \sum_i x_i y_i$. Thus, a kernel method is an algorithm that can be written down in such a way that all data vectors appear within a scalar product operation. To "kernelize" the algorithm, we then simply replace the scalar product operation with the kernel function $K$.

Substituting the kernel function for the scalar product operation is useful because it is mathematically equivalent to projecting the data set into a different space. Say that the input space has $m$ dimensions, but we use a quadratic kernel function defined as $K(x, y) = (\sum_i x_i y_i)^2$. In this case, we are implicitly working in a space of

$m^2$ dimensions. This higher-dimensional space is called the *feature space*, and in this example, it contains one dimension for every pair of dimensions in the input space. This kernel can thus capture pairwise correlations between input variables. A kernelized version of the SVM algorithm finds the maximum margin hyperplane in the feature space, simply by solving the original optimization problem using a different kernel function.

### 2.2.3 Combining kernels

Often, kernels are useful because they allow the SVM to find a separating hyperplane in a data set that was previously inseparable. Kernels may also allow us to encode prior knowledge about the data, such as the knowledge the pairwise correlations are important. In the current work, however, we are particularly interested in the kernel function as a way to encode similarities among non-vector and heterogeneous data sets.

First, we note that although the discussion thus far has focused on vector data, a kernel function can be defined for any arbitrarily complex data object. Thus, kernels have been defined for DNA and protein sequences, protein-protein and metabolic networks, phylogenetic trees, etc. [Noble, 2004] As long as our collection of data can be represented as a square kernel matrix, then any kernel method can be applied to the data. The kernel matrix is thus a sort of normal form for representing diverse types of data.

Second, the mathematics of kernels allows us to derive new kernels by combining two or more kernel functions. Many mathematical operations are closed under the kernel property. The most important such operation is addition: if $K_1$ and $K_2$ are both kernel functions, then we can prove that $K(x, y) = K_1(x, y) + K_2(x, y)$ is a kernel. This operation is mathematically equivalent to concatenating the vector representations of the two data points in the feature spaces defined by $K_1$ and $K_2$.

This mathematical formalism provides us with a straightforward way to combine

heterogeneous data. Given a set of proteins represented as sequences and structures, we compute a kernel matrix $K_q$ from the sequences and a kernel matrix $K_r$ from the structures. The sum of these two kernel matrices is a new kernel that simultaneously represents the protein sequences and structures. As mentioned in the introduction, this simple sum-of-kernels approach has been used successfully, e.g., to predict protein-protein interactions using a combination of protein sequences, functional annotation, and known protein-protein interactions [Ben-Hur and Noble, 2005].

In the current work, we contrast the simple sum-of-kernels approach with a more sophisticated method that introduces weights on each kernel. Using one of various optimization methods, we can simultaneously find a separating hyperplane and find weights on each individual kernel. The weights are chosen so as to maximize the margin between the two classes. This approach corresponds to learning, e.g., that the sequence kernel is not as informative as the structure kernel for a given classification task. Geometrically, the kernel weight re-scales each dimension of a given kernel. Thus, in the feature space corresponding to $K = K_1 + 2K_2$, each of the dimensions from $K_2$ is scaled by a factor of 2.

## 2.3 Methods

### 2.3.1 Gene Ontology classes

The Gene Ontology [Gene Ontology Consortium, 2000] is a diverse catalog of gene (protein) annotation, which includes information about protein function and localization. We define a GO term prediction benchmark by starting with a set of 8363 PDB structures, pruned so that no two sequences share greater than 50% sequence identity [Li et al., 2001]. Among these proteins, 5325 have GO annotations, downloaded from `www.geneontology.org`. For each GO term $T$, we partitioned the list of proteins into three sets. First, all proteins that are annotated with $T$ are labeled as "positive." Next, we traverse from $T$ along all paths to the root of the Gene Ontology graph. At

each GO term along this path, we look for proteins that are assigned to that term and not to any of that term's children. We consider that such proteins might be properly assigned to $T$, and so we label those proteins as "uncertain." Finally, all proteins that are not on the path from $T$ to the root are labeled as "negative."

After this labeling procedure, we eliminated all GO terms with fewer than 100 "positive" proteins. In order to avoid redundancy, we then selected only the most specific of the remaining GO terms, i.e., the leaf nodes of the remaining hierarchy. This procedure yielded a total of 166 GO terms: 27 molecular function terms, 22 biological process terms, and 7 cellular component terms. All 56 terms are listed in the supplementary results (Section 2.6).

For each GO term, the number of negative examples far exceeds the number of positive examples. For efficiency, we randomly select a subset of the negative examples so that the ratio of positives to negatives is one-to-one.

### 2.3.2 Kernels

To represent protein sequences, we use the mismatch kernel [Leslie et al., 2003]. This kernel generalizes upon the simpler, spectrum kernel [Leslie et al., 2002], which represents a string as a vector of counts of all possible substrings of a fixed length $k$. The mismatch kernel generalizes upon the spectrum kernel by incrementing, for each observed $k$-length string ($k$-mer), the corresponding count as well as the counts of all $k$-mers that differ from the observed $k$-mer by at most $m$ mismatches. In this work, we use a mismatch spectrum kernel with $k = 4$ and $m = 1$. The final mismatch spectrum vector has $20^4 = 160,000$ bins. The mismatch spectrum captures sequence similarity, and has been shown to provide good performance in classifying SCOP superfamilies [Leslie et al., 2003].

Insofar as a kernel function defines the similarity between pairs of objects, the most natural place to begin defining a protein structure kernel is with existing pairwise structure comparison algorithms. Many such algorithms exist, including CE

[Shindyalov and Bourne, 1998], DALI [Holm and Sander, 1993] and MAMMOTH [Ortiz et al., 2002]. Most of these algorithms attempt to create an alignment between two proteins and then compute a score that reflects the alignment's quality. In this work, we use MAMMOTH [Ortiz et al., 2002], which is efficient and produces high quality alignments.

Unfortunately, the alignment quality score returned by MAMMOTH cannot be used as a kernel function directly, because the score is not positive semi-definite (i.e., for a given set of protein structures, an all-versus-all matrix of MAMMOTH scores will have some negative eigenvalues). We therefore employ the so-called "empirical kernel map" [Tsuda, 1999] to convert this score to a kernel: for a given data set of structures $X = x_1, \ldots x_n$, a structure $x_i$ is represented as an $n$-dimensional vector, in which the $j$th entry is the MAMMOTH score between $x_i$ and $x_j$. The SVM then uses this vector representation directly. This method has been used successfully in the SVM-pairwise method of remote protein homology detection [Liao and Noble, 2002], in which a protein is represented as a vector of log E-values from a pairwise sequence comparison algorithm such as Smith-Waterman [Smith and Waterman, 1981]. In our experiments, we use the log of the E-value returned by MAMMOTH. The resulting MAMMOTH kernel incorporates information about the alignability of a given pair of proteins.

### 2.3.3 Experimental framework

All SVM experiments were performed using our own code, which is a combination of C++ and Matlab. To compute weighted kernel combinations we use semidefinite programming, as described in Lanckriet et al. [2002]. SVMs were tested using five-fold cross-validation, repeated three times (3x5cv). We use a fixed value of the SVM regularization parameter $C = 10$. We measure classification performance using the area under the receiver operating characteristic (ROC) curve, which plots the rate of true positives as a function of the rate of false positives for varying classification

thresholds. We report means and standard deviations of the ROC with respect to the fifteen 3x5cv splits.

## 2.4 Results

We present our results as a series of four experiments. The first experiment is a direct comparison of the unweighted and weighted kernel combination methods across all 56 GO terms in our benchmark. The results show that the structure kernel consistently out-performs the sequence kernel, and that the unweighted combination generally performs better than the weighted combination. In the second experiment, we systematically vary the relative kernel weight on a subset of 10 GO terms, and we show that, for this task, the unweighted sum of kernels performs nearly optimally. The third experiment introduces artificial noise into the data set. In this scenario, the weighted kernel approach is useful, and performs better than the unweighted approach as the amount of noise increases. Finally, in a fourth experiment, we demonstrate the robustness of both kernel combination methods in the presence of missing data.

### 2.4.1 Experiment 1: Comparison of kernel combination methods

In this experiment, we performed cross-validated testing of four types of SVMs, using sequence alone, structure alone, an unweighted combination of kernels, and a weighted combination of kernels. Table 2.1 shows a subset of these results. The complete set of results are available in the supplementary results (Section 2.6). Not surprisingly, the MAMMOTH kernel frequently provides better classification performance: in 55 out of 56 classes, the difference between the mean structure ROC and the mean sequence ROC is greater than the sum of the two corresponding standard errors.

An alternate representation of these results, for all 56 terms, is shown in Figure 2.1. Qualitatively, the figure shows that the sequence kernel performs far worse than

Table 2.1: **Predicting GO terms from sequence or from structure.** Each row in the table lists a GO term and description, the ontology from which it comes (MF = molecular function, CC = cellular compartment and BP = biological process), the number of positive examples associated with the term, the mean and standard error ROC scores for 3x5cv SVM training using (1) the structure kernel, (2) the sequence kernel, (3) the unweighted sum of both kernels, and (4) the weighted sum of the kernel. In the table, terms are sorted by the difference in ROC between "Structure" and "Sequence." From the entire set of 56 terms that we considered, the table lists only the top 10 and the bottom 5.

| GO term | Description | Ont | # | Structure | Sequence | Average | SDP |
|---|---|---|---|---|---|---|---|
| GO:0008168 | methyltransferase activity | MF | 108 | 0.941 ± 0.014 | 0.709 ± 0.020 | 0.937 ± 0.016 | 0.938 ± 0.015 |
| GO:0005506 | iron ion binding | MF | 129 | 0.934 ± 0.008 | 0.747 ± 0.015 | 0.927 ± 0.012 | 0.927 ± 0.012 |
| GO:0006260 | DNA replication | BP | 109 | 0.885 ± 0.014 | 0.707 ± 0.020 | 0.878 ± 0.016 | 0.870 ± 0.015 |
| GO:0048037 | cofactor binding | MF | 118 | 0.916 ± 0.015 | 0.738 ± 0.025 | 0.911 ± 0.016 | 0.909 ± 0.016 |
| GO:0046483 | heterocycle metabolism | BP | 128 | 0.949 ± 0.007 | 0.787 ± 0.011 | 0.937 ± 0.008 | 0.940 ± 0.008 |
| GO:0044255 | cellular lipid metabolism | BP | 101 | 0.891 ± 0.012 | 0.732 ± 0.012 | 0.874 ± 0.015 | 0.864 ± 0.013 |
| GO:0016853 | isomerase activity | MF | 124 | 0.855 ± 0.014 | 0.706 ± 0.029 | 0.837 ± 0.017 | 0.810 ± 0.019 |
| GO:0044262 | cellular carbohydrate metabolism | BP | 209 | 0.912 ± 0.007 | 0.764 ± 0.018 | 0.908 ± 0.006 | 0.897 ± 0.006 |
| GO:0009117 | nucleotide metabolism | BP | 124 | 0.892 ± 0.015 | 0.748 ± 0.016 | 0.890 ± 0.012 | 0.880 ± 0.012 |
| GO:0016829 | lyase activity | MF | 201 | 0.935 ± 0.006 | 0.791 ± 0.013 | 0.931 ± 0.008 | 0.926 ± 0.007 |
| GO:0006732 | coenzyme metabolism | BP | 119 | 0.823 ± 0.011 | 0.781 ± 0.013 | 0.845 ± 0.011 | 0.828 ± 0.013 |
| GO:0007242 | intracellular signaling cascade | BP | 140 | 0.898 ± 0.011 | 0.859 ± 0.014 | 0.903 ± 0.010 | 0.900 ± 0.011 |
| GO:0005525 | GTP binding | MF | 104 | 0.923 ± 0.008 | 0.884 ± 0.015 | 0.931 ± 0.009 | 0.931 ± 0.009 |
| GO:0004252 | serine-type endopeptidase activity | MF | 140 | 0.937 ± 0.011 | 0.907 ± 0.012 | 0.932 ± 0.012 | 0.931 ± 0.012 |
| GO:0005198 | structural molecule activity | MF | 179 | 0.809 ± 0.010 | 0.795 ± 0.014 | 0.828 ± 0.010 | 0.824 ± 0.011 |

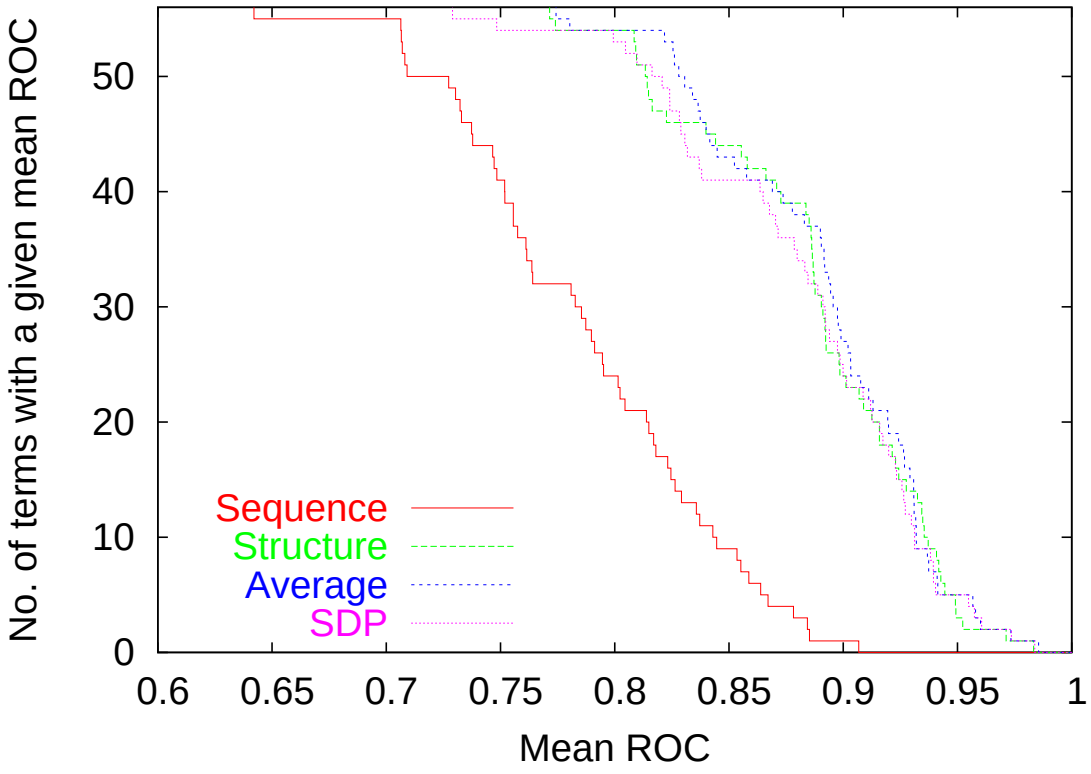


Figure 2.1: **Combining kernels: cumulative comparison across 56 GO terms.** The figure plots the number of GO terms (y-axis) for which a given SVM classifier achieves a specified mean ROC score (x-axis). Each series corresponds to an SVM that uses sequence alone, structure alone, or combinations of kernels using an unweighted average or using SDP.

any method that uses the structure kernel. Furthermore, we computed a Wilcoxon signed-rank test between all four pairs of methods. The results yield the following best-to-worst ranking of methods: unweighted sum of kernels, structure kernel alone, weighted sum of kernels, and sequence kernel alone. In this ranking, the largest (i.e., least significant) p-value is 0.007 between the structure kernel alone and the weighted sum of kernels. Thus it appears that combining sequence and structure can be helpful, but only when using the unweighted sum of kernels.

### 2.4.2 Experiment 2: Varying relative kernel weight

The previous result—that the weighted sum of kernels performs worse than the unweighted sum—is surprising, not least because considerable effort has been expended by various research groups to develop the optimization technology to solve this type of MKL problem. Therefore, we selected a subset of the terms from our benchmark and subjected them to further investigation. Specifically, we selected the 10 terms for which the difference in ROC between the structure-only and sequence-only SVMs is largest. For each of these terms, we systematically varied the relative kernel weights, and performed cross-validated testing of the resulting SVM. The results are shown in Figure 2.2.

The most striking aspect of Figure 2.2 is the qualitative similarity of all ten series in the figure. For each GO term, the performance of the SVM stays roughly the same for all kernel combinations that assign larger weight to the structure kernel. When more weight is assigned to the sequence kernel, the performance degrades gradually, with a rapid degradation only when the structure kernel receives very small weight. This result shows that the SVM is quite robust in the presence of variation in the relative scales of the two data sets.

A second observation is the placement of the green dots, indicating the choice of kernel weights made by the SDP optimization. In nearly every case, the SDP erroneously gives more weight to the sequence kernel. This explains why the weighted

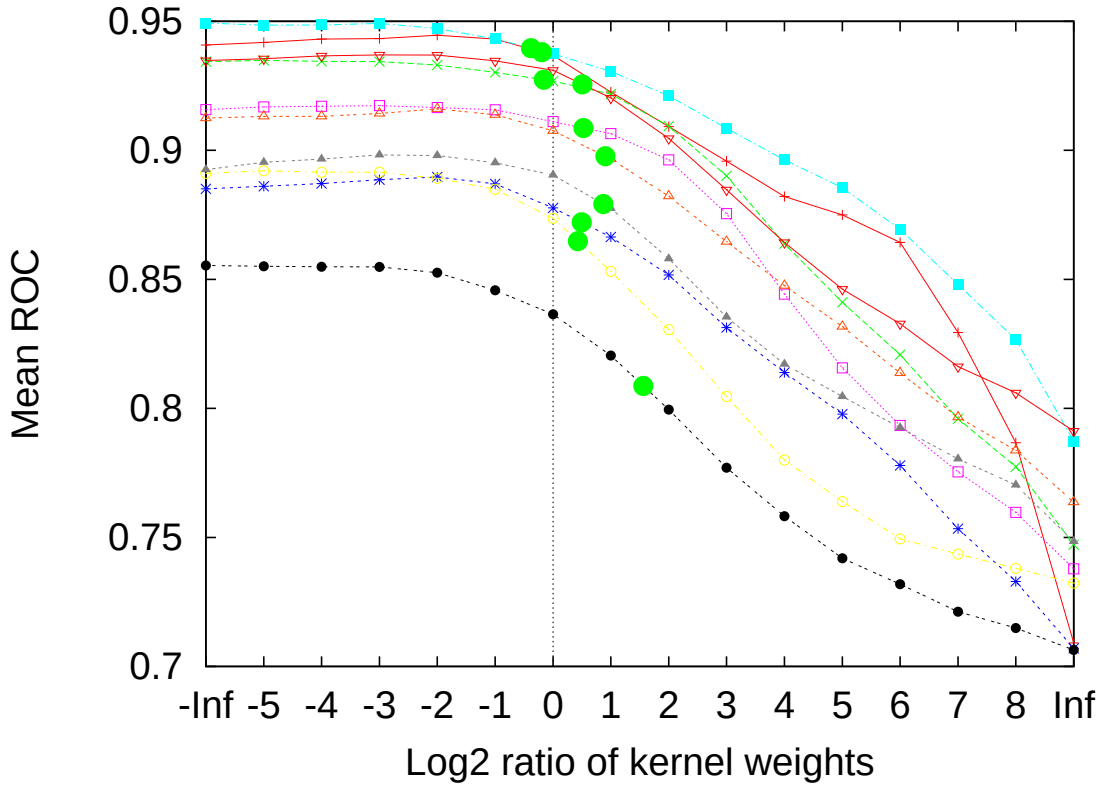


Figure 2.2: **Varying relative kernel weight.** The figure plots mean ROC score as a function of the $\log_2(\mu_q/\mu_r)$, where $\mu_q$ and $\mu_r$ are the weights assigned to the sequence and structure kernels, respectively. Each series corresponds to one of the GO terms in Table 2.1. On each series, the large green circle indicates the log ratio of the weights selected by SDP.

kernel combination fares poorly on this benchmark. Apparently, for these tasks, the sequence kernel yields an embedding with a larger margin than does the structure kernel, even though the latter provides better generalization performance.

Finally, it is interesting to consider whether the unweighted sum of kernels is optimal. For many of the GO terms in Figure 2.2, the highest ROC is not achieved at a log ratio of 0. However, in every case, the difference between the unweighted sum ROC and the best possible ROC is quite small, on the order of 0.01. Furthermore, for any fixed log ratio, there are some terms that perform worse than the unweighted average and some that perform better. Thus, Figure 2.2 suggests that although an optimal learning procedure might be able to find better kernel weights than the unweighted average, this hypothetical method (1) would have to take into consideration not just the kernel matrices but also the GO term labels, and (2) would still perform only slightly better than the unweighted sum of kernels.

### 2.4.3 Experiment 3: Multiple noisy kernels

The results from our first two experiments beg the question: why bother with the computational overhead of weighted kernel combinations when the unweighted sum of kernels performs better? In the third experiment, we demonstrate via simulation a situation in which the weighted sum is necessary.

To carry out this experiment, we created a collection of "noise" kernels. These are simply copies of the structure kernel, with permuted rows and corresponding columns. The resulting matrices are still kernels, but the entries no longer correspond to the labeling. We then measured how the performance of our two kernel combination methods changes as the number of noise kernels increases.

The results are shown in Figure 2.3. In the figure, green crosses correspond to classifiers trained with no noise kernels. These points were thus generated in our first experiment, described above. Most of the points fall below the line $y = x$, indicating clearly that the unweighted kernel combination performs better than the weighted

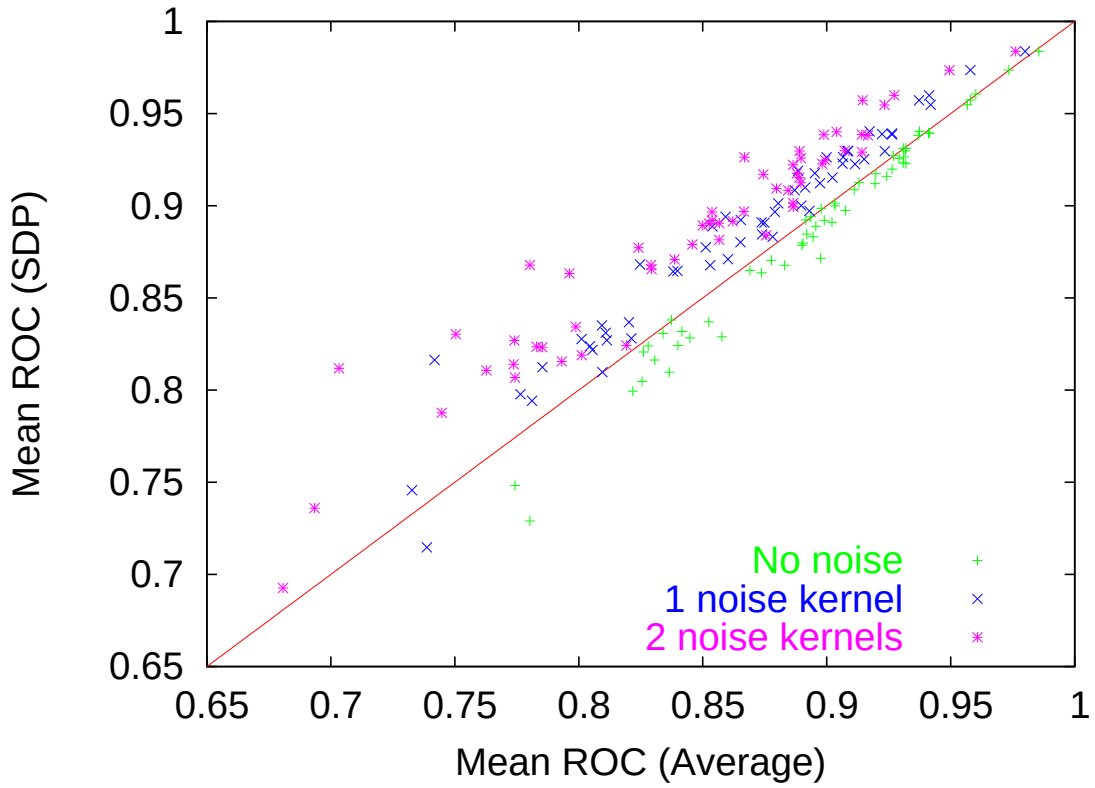


Figure 2.3: **Learning in the presence of noisy kernels.** The figure plots, for each GO term, the ROC score achieved by the SDP SVM as a function of the ROC score achieved by the unweighted sum of kernels. Different point types correspond to training using zero, one or two noise kernels, as described in the text.

sum of kernels. However, this result changes as soon as we introduce a single noise kernel. In this case, the weighted sum performs better than the unweighted sum for 55 out of the 56 terms. The effect becomes even more pronounced in the presence of two noise kernels. A Wilcoxon signed-rank test supports these conclusions with extremely small $p$-values.

### 2.4.4 Experiment 4: Kernels with missing examples

Finally, we consider a variant of our experimental design, in which we focus on the problem of missing data. In particular, we are interested in the extent to which we can combine incomplete structural information with complete sequence information. Thus, we simulate randomly deleting varying percentages of the examples from the structure kernel matrix, and measure the cross-validated ROC score of the resulting classifier. A pseudocode description of the experimental design is given in the supplementary results (Section 2.6).

Because SVMs are not designed to handle missing data, it is not obvious *a priori* how to represent missing examples in the kernel matrix. We therefore consider three alternative methods for filling in missing kernel entries. The first strategy, "None," simply replaces the row and column corresponding to a missing entry with all zeroes. This strategy allows the sequence kernel to determine the weight assigned by the SVM to this example, ignoring the structure kernel entirely. The second strategy ("Self") makes each missing example similar only to itself by placing a 1 on the diagonal of the (normalized) kernel matrix and zeroes elsewhere. Finally, the "All" strategy makes each missing example similar to every other missing example, but different from all non-missing examples. This is accomplished by placing 1s in the kernel matrix between all pairs of missing examples, and 0s between missing/non-missing pairs. The effect is to place all missing examples in a single orthogonal dimension in feature space. Effectively, all missing examples are co-located at a single point, infinitely distant from the other data.

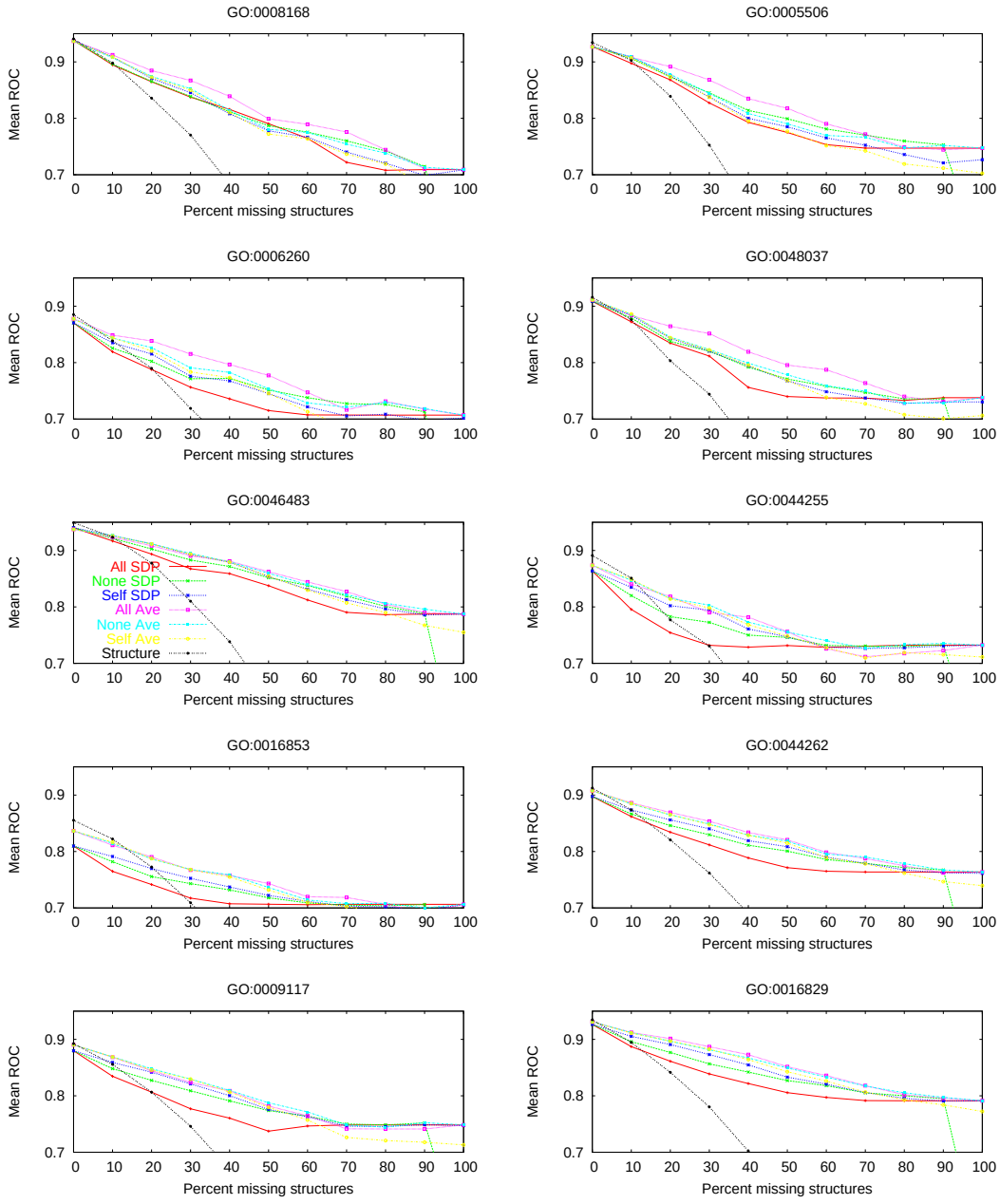


Figure 2.4: **Combining kernels with missing data.** Each figure plots, for a single GO term, the mean ROC score as a function of the percentage of missing data in the structure kernel. The first six series correspond to the AVE and SDP methods, with missing data affinity coded as "None," "Self" or "All." The final series is from an SVM trained from the structure kernel alone.

Figure 2.4 shows the results of this experiment. Strikingly, none of the series deteriorates dramatically as we introduce missing data. Indeed, most methods perform better than the sequence kernel alone even when the structure kernel consists of 50% missing entries.

Trends among the six methods that we considered—two kernel combination methods and three methods for replacing missing values—are not obvious from Figure 2.4. A Wilcoxon signed-rank comparison of the data (see supplementary results (Section 2.6)) yields the same ranking of methods at 10% or 20% missing data: all three unweighted sum methods perform better than all three weighted sum methods, and the best strategy for handling missing data depends upon the kernel combination method. Using an unweighted combination, the best-to-worst ranking is All-None-Self. Conversely, using a weighted combination, the corresponding ranking is Self-None-All.

In short, this experiment does show convincingly that an SVM can make accurate predictions in the presence of missing data; however, the results are inconclusive with respect to the best method for representing missing examples in the kernel matrices.

## 2.5 Discussion

The primary conclusion from this empirical study is that using a weighted sum of kernels in an SVM classifier does not always improve upon the simpler, unweighted sum approach. In particular, we have shown that, for a combination of sequence and structure kernels in the prediction of Gene Ontology terms, the weighted sum method frequently selects a solution that is worse than the unweighted sum. On the other hand, the weighted sum does appear to improve the SVM's robustness in the presence of noisy or irrelevant kernels. From a practitioner's point of view, these results suggest the value of evaluating individual kernels with respect to any given classification task prior to applying a kernel combination method. In other words,

simply collecting a large variety of kernels and applying the resulting combination of kernels to diverse classification tasks is not likely to be as successful as a more directed approach, in which prior knowledge of a particular kernel's relevance guides its inclusion in the training set.

Our results also suggest that the kernel weights assigned by an MKL method may be difficult to interpret. *A priori*, it is clear that a low weight may be assigned due either to noise in the kernel or to redundancy with another kernel in the collection. However, for many of the classification tasks that we considered, a relatively broad range of relative kernel weights often yielded quite similar classification performance. Furthermore, in this particular case, the size of the margin does not correlate well with optimal generalization performance.

With respect to protein classification, it is perhaps not surprising that structure is more informative than sequence. However, our results with respect to missing data suggest that, even when some structure data is not present, an SVM combination of sequence and structure might be valuable.

The lower performance of the weighted kernel combination might be due to overfitting and insufficient training data. Thus, larger data sets might benefit from weighted combination when there is sufficient data to reliably estimate kernel weights. Alternatively, we might group multiple tasks and datasets to more reliably estimate a single setting of the weights on kernels [Jebara, 2004b].

Any empirical study necessarily leaves some questions unanswered: one could imagine a variety of modifications, extensions or additional experiments to add to the four we described above. These include, for example, investigating different types or a larger number of kernels, modifying the SVM regularization parameter, systematically adding noise to one or more kernels, etc. While some of these experiments would likely be more informative than others, we believe that our primary message—that combining kernels in a weighted fashion is not always beneficial—would remain unchanged. Further experiments would more precisely define the situations in which

weighting kernels is beneficial.

We did, inadvertently, perform one additional experiment that was not reported above, and we believe that the result is instructive. In setting up the experiment comparing kernel combination methods across 56 GO terms, we first observed that the SDP method almost uniformly gave a large weight to the sequence kernel and a small weight to the structure kernel. Investigation of these results showed that the sequence kernel margin was considerably larger simply because we had normalized the kernels (i.e., projected all of the data onto the unit sphere in the feature space) without first centering the data around the origin. In both cases, the data were far from the origin, and so projection placed all of the data points onto a very small area on the unit sphere. Centering before normalization, as recommended by Lanckriet et al. [2002], leads to much better conditioned matrices. This result illustrates that the weighted kernel method depends upon characteristics of the various kernels in the combination. One avenue for future work would identify characteristics of "good" kernels, and propose methods for creating such kernels.

A second area for future work is the development of alternate kernel combination methods. For example, given the relatively robust performance of SVMs with respect to gradations in relative kernel weight, we believe that a combinatorial method which finds binary kernel weights might be very successful. Unfortunately, it is not obvious how to perform efficient optimization on discrete kernel weights.

## 2.6 Supplementary Results

**Algorithm 1** Cross-validation framework for the missing data experiment. The input variables are defined as follows: $L$ = label matrix, $Q$ = sequence kernel matrix, $R$ = structure kernel matrix, $\mathcal{C}$ = set of classes, $I$ = number of iterations, $S$ = number of cross-validation splits, $\mathcal{P}$ = set of percentages missing, $\mathcal{M}$ = set of supervised learning methods. The output of the algorithm is a matrix $\mathbf{R}$ of mean ROC scores, indexed by class, percentage of missing data, and learning method.

**procedure** MISSING DATA EXPERIMENT($L, Q, R, \mathcal{C}, I, S, \mathcal{P}, \mathcal{M}$)
  $\mathbf{R} \leftarrow [0, \ldots, 0]$ ▷ Initialize all ROC scores to zero.
  **for** $c \in \mathcal{C}$ **do** ▷ Begin the main experiment.
    $\ell \leftarrow$ selectClass($L, c$)
    **for** $i \leftarrow 1 \ldots I$ **do**
      **for** $s \leftarrow 1 \ldots S$ **do**
        $(\ell_R, \ell_E, Q_R, Q_E, R_R, R_E) \leftarrow$ cvSplit($\ell, Q, R, s, S$) ▷ Split into train and test sets.
        **for** $p \in \mathcal{P}$ **do**
          $R'_R \leftarrow$ makeMissing($p, R_R$) ▷ Insert missing rows and columns.
          $R'_E \leftarrow$ makeMissing($p, R_E$)
          **for** $m \in \mathcal{M}$ **do**
            classifier $\leftarrow$ train($m, \ell_R, Q_R, R'_R$) ▷ Train the classifier.
            predictions $\leftarrow$ test(classifier, $Q_E, R'_E$) ▷ Make predictions on the test set.
            $\mathbf{R}_{c,p,m} \leftarrow \mathbf{R}_{c,p,m}+$ computeRoc($\ell_E$, predictions) $/(S * I)$ ▷ Score the predictions.
          **end for**
        **end for**
      **end for**
    **end for**
  **end for**
**end procedure**

Table 2.2: **Predicting GO terms from sequence or from structure.** Results for the 41 remaining GO terms not shown in Table 2.1.

| GO term | Description | Ont | # | Structure | Sequence | Average | SDP |
|---|---|---|---|---|---|---|---|
| GO:0016779 | nucleotidyltransferase activity | MF | 142 | 0.866 ± 0.013 | 0.727 ± 0.020 | 0.869 ± 0.016 | 0.865 ± 0.017 |
| GO:0016043 | cell organization and biogenesis | BP | 106 | 0.871 ± 0.013 | 0.733 ± 0.018 | 0.858 ± 0.013 | 0.829 ± 0.014 |
| GO:0008270 | zinc ion binding | MF | 234 | 0.892 ± 0.006 | 0.756 ± 0.016 | 0.896 ± 0.010 | 0.889 ± 0.011 |
| GO:0006066 | alcohol metabolism | BP | 111 | 0.921 ± 0.010 | 0.785 ± 0.025 | 0.919 ± 0.010 | 0.912 ± 0.010 |
| GO:0003723 | RNA binding | MF | 212 | 0.886 ± 0.011 | 0.752 ± 0.014 | 0.883 ± 0.011 | 0.868 ± 0.010 |
| GO:0004518 | nuclease activity | MF | 125 | 0.840 ± 0.017 | 0.708 ± 0.016 | 0.831 ± 0.016 | 0.816 ± 0.014 |
| GO:0006811 | ion transport | BP | 117 | 0.771 ± 0.014 | 0.642 ± 0.020 | 0.774 ± 0.016 | 0.748 ± 0.017 |
| GO:0006725 | aromatic compound metabolism | BP | 164 | 0.890 ± 0.008 | 0.761 ± 0.014 | 0.894 ± 0.008 | 0.883 ± 0.009 |
| GO:0016491 | oxidoreductase activity | MF | 516 | 0.952 ± 0.002 | 0.823 ± 0.007 | 0.957 ± 0.003 | 0.955 ± 0.003 |
| GO:0009405 | pathogenesis | BP | 118 | 0.942 ± 0.008 | 0.814 ± 0.017 | 0.941 ± 0.008 | 0.939 ± 0.008 |
| GO:0005524 | ATP binding | MF | 485 | 0.888 ± 0.006 | 0.764 ± 0.008 | 0.899 ± 0.005 | 0.892 ± 0.005 |
| GO:0030246 | carbohydrate binding | MF | 102 | 0.924 ± 0.008 | 0.801 ± 0.017 | 0.929 ± 0.007 | 0.926 ± 0.008 |
| GO:0006508 | proteolysis and peptidolysis | BP | 330 | 0.945 ± 0.003 | 0.829 ± 0.009 | 0.941 ± 0.004 | 0.940 ± 0.004 |
| GO:0008652 | amino acid biosynthesis | BP | 121 | 0.916 ± 0.013 | 0.802 ± 0.017 | 0.924 ± 0.009 | 0.916 ± 0.009 |
| GO:0045184 | establishment of protein localization | BP | 108 | 0.844 ± 0.017 | 0.737 ± 0.014 | 0.842 ± 0.015 | 0.832 ± 0.013 |
| GO:0020037 | heme binding | MF | 104 | 0.983 ± 0.006 | 0.878 ± 0.015 | 0.986 ± 0.005 | 0.984 ± 0.006 |
| GO:0003700 | transcription factor activity | MF | 214 | 0.932 ± 0.007 | 0.836 ± 0.008 | 0.932 ± 0.007 | 0.923 ± 0.007 |
| GO:0016070 | RNA metabolism | BP | 140 | 0.886 ± 0.009 | 0.795 ± 0.018 | 0.894 ± 0.010 | 0.894 ± 0.010 |
| GO:0005102 | receptor binding | MF | 120 | 0.909 ± 0.010 | 0.818 ± 0.018 | 0.932 ± 0.008 | 0.930 ± 0.008 |
| GO:0006355 | regulation of transcription, DNA-dependent | BP | 340 | 0.928 ± 0.006 | 0.837 ± 0.010 | 0.931 ± 0.007 | 0.923 ± 0.008 |
| GO:0016874 | ligase activity | MF | 161 | 0.873 ± 0.011 | 0.783 ± 0.010 | 0.892 ± 0.009 | 0.892 ± 0.009 |
| GO:0006468 | protein amino acid phosphorylation | BP | 160 | 0.892 ± 0.010 | 0.804 ± 0.016 | 0.890 ± 0.009 | 0.879 ± 0.010 |
| GO:0016798 | hydrolase activity, acting on glycosyl bonds | MF | 227 | 0.971 ± 0.005 | 0.885 ± 0.010 | 0.973 ± 0.005 | 0.974 ± 0.005 |
| GO:0006118 | electron transport | BP | 392 | 0.949 ± 0.003 | 0.864 ± 0.006 | 0.960 ± 0.003 | 0.961 ± 0.003 |
| GO:0004672 | protein kinase activity | MF | 164 | 0.898 ± 0.009 | 0.817 ± 0.014 | 0.892 ± 0.009 | 0.885 ± 0.010 |
| GO:0004872 | receptor activity | MF | 138 | 0.935 ± 0.007 | 0.855 ± 0.015 | 0.926 ± 0.006 | 0.920 ± 0.006 |
| GO:0015075 | ion transporter activity | MF | 110 | 0.808 ± 0.020 | 0.730 ± 0.019 | 0.822 ± 0.018 | 0.799 ± 0.016 |
| GO:0005489 | electron transporter activity | MF | 196 | 0.943 ± 0.008 | 0.867 ± 0.009 | 0.958 ± 0.007 | 0.957 ± 0.007 |
| GO:0005576 | extracellular region | CC | 352 | 0.887 ± 0.007 | 0.815 ± 0.009 | 0.897 ± 0.006 | 0.872 ± 0.007 |
| GO:0019012 | virion | CC | 101 | 0.858 ± 0.015 | 0.790 ± 0.027 | 0.852 ± 0.016 | 0.837 ± 0.017 |
| GO:0030234 | enzyme regulator activity | MF | 132 | 0.815 ± 0.010 | 0.747 ± 0.016 | 0.826 ± 0.009 | 0.805 ± 0.011 |
| GO:0016021 | integral to membrane | CC | 136 | 0.774 ± 0.009 | 0.707 ± 0.019 | 0.780 ± 0.008 | 0.729 ± 0.018 |
| GO:0006412 | protein biosynthesis | BP | 170 | 0.886 ± 0.010 | 0.825 ± 0.009 | 0.898 ± 0.009 | 0.899 ± 0.009 |
| GO:0005634 | nucleus | CC | 347 | 0.901 ± 0.006 | 0.843 ± 0.007 | 0.920 ± 0.005 | 0.917 ± 0.005 |
| GO:0017111 | nucleoside-triphosphatase activity | MF | 154 | 0.813 ± 0.013 | 0.756 ± 0.017 | 0.826 ± 0.015 | 0.821 ± 0.015 |
| GO:0005737 | cytoplasm | CC | 490 | 0.884 ± 0.006 | 0.826 ± 0.006 | 0.902 ± 0.005 | 0.891 ± 0.005 |
| GO:0051188 | cofactor biosynthesis | BP | 118 | 0.809 ± 0.017 | 0.752 ± 0.019 | 0.834 ± 0.016 | 0.831 ± 0.017 |
| GO:0043232 | intracellular non-membrane-bound organelle | CC | 153 | 0.814 ± 0.009 | 0.757 ± 0.013 | 0.837 ± 0.012 | 0.838 ± 0.015 |
| GO:0043234 | protein complex | CC | 414 | 0.816 ± 0.009 | 0.761 ± 0.009 | 0.840 ± 0.009 | 0.824 ± 0.009 |
| GO:0005509 | calcium ion binding | MF | 173 | 0.907 ± 0.011 | 0.854 ± 0.009 | 0.913 ± 0.011 | 0.913 ± 0.010 |
| GO:0050874 | organismal physiological process | BP | 144 | 0.887 ± 0.010 | 0.845 ± 0.014 | 0.903 ± 0.010 | 0.901 ± 0.010 |

Table 2.3: **Signed rank comparisons for Experiment 1.** The table lists uncorrected $p$-values from a Wilcoxon signed-rank test comparison of the results of Experiment 1. A significant $p$-value in the table indicates that method along the row performs significantly better than the method along the column.

| | Sequence | Average | SDP | Structure |
|---|---|---|---|---|
| Sequence | — | — | — | — |
| Average | 1.078e-134 | — | 7.519e-49 | 4.038e-09 |
| SDP | 4.649e-133 | — | — | — |
| Structure | 6.457e-128 | — | 0.00667 | — |

Table 2.4: **Signed rank comparisons for Experiment 3.** The table lists uncorrected $p$-values from a Wilcoxon signed-rank test comparison of the results from Experiment 3. A significant $p$-value in the table indicates that method along the row performs significantly better than the method along the column.

| | Average | SDP | Average 1-Noise | SDP 1-Noise | Average 2-Noise | SDP 2-Noise |
|---|---|---|---|---|---|---|
| Average | — | 7.519e-49 | 5.845e-118 | 7.319e-55 | 9.221e-129 | 9.538e-59 |
| SDP | — | — | 1.415e-84 | 0.000287 | 1.618e-113 | 1.115e-09 |
| Average 1-Noise | — | — | — | — | 2.499e-70 | — |
| SDP 1-Noise | — | 0.000287 | 2.372e-83 | — | 2.857e-114 | 1.633e-05 |
| Average 2-Noise | — | — | — | — | — | — |
| SDP 2-Noise | — | 1.115e-09 | 1.726e-70 | 1.633e-05 | 7.573e-116 | — |

Table 2.5: **Signed rank comparisons for Experiment 4.** Each table lists, for a given percent missing, uncorrected $p$-values from a Wilcoxon signed-rank test comparison of the six classification methods in Experiment 4. A significant $p$-value in the table indicates that method along the row performs significantly better than the method along the column.

| 10% | Average (All) | SDP (All) | Average (Self) | SDP (Self) | Average (None) | SDP (None) |
|---|---|---|---|---|---|---|
| Average (All) | — | 4.925e-75 | 5.483e-31 | 2.994e-55 | 9.502e-30 | 2.649e-61 |
| SDP (All) | — | — | — | — | — | — |
| Average (Self) | — | 1.02e-16 | — | 6.087e-17 | — | 3.767e-22 |
| SDP (Self) | — | 4.951e-10 | — | — | — | 1.513e-19 |
| Average (None) | — | 3.92e-19 | 3.894e-05 | 3.832e-23 | — | 8.638e-29 |
| SDP (None) | — | 0.001914 | — | — | — | — |
| 20% | Average (All) | SDP (All) | Average (Self) | SDP (Self) | Average (None) | SDP (None) |
| Average (All) | — | 8.145e-75 | 3.438e-58 | 6.751e-61 | 1.112e-47 | 1.946e-57 |
| SDP (All) | — | — | — | — | — | — |
| Average (Self) | — | 4.444e-10 | — | — | — | 0.0007737 |
| SDP (Self) | — | 3.895e-14 | — | — | — | 1.864e-07 |
| Average (None) | — | 1.695e-16 | 4.963e-22 | 8.506e-12 | — | 9.056e-11 |
| SDP (None) | — | 1.569e-12 | — | — | — | — |
| 30% | Average (All) | SDP (All) | Average (Self) | SDP (Self) | Average (None) | SDP (None) |
| Average (All) | — | 7.182e-66 | 1.76e-64 | 1.718e-58 | 8.461e-50 | 2.364e-50 |
| SDP (All) | — | — | — | — | — | — |
| Average (Self) | — | 1.801e-05 | — | — | — | — |
| SDP (Self) | — | 1.157e-13 | 0.0003909 | — | — | — |
| Average (None) | — | 4.116e-14 | 5.129e-37 | 1.805e-07 | — | 0.0008848 |
| SDP (None) | — | 8.058e-22 | 0.04355 | — | — | — |

# Chapter 3

# Nonstationary Kernel Combination

This chapter presents our primary contribution to the machine learning community. The work presented herein has been accepted, in abridged form, to the *International Conference on Machine Learning (2006)* [Lewis, Jebara, and Noble, 2006b], and the full text has been submitted to the *Journal of Machine Learning Research* [Lewis, Jebara, and Noble, 2006a] along with the empirical results of Chapter 4.

The technique of maximum entropy discrimination was introduced by Jaakkola, Meila, and Jebara [1999c] and was extended in Jebara [2004a]. Here, we build on the prior work to present an improved latent MED variational bound, a latent MED mixture of Gaussians for multi-kernel learning, and an efficient optimization procedure. We then present the technique as a nonstationary kernel combination, by deriving the combination weights. We show that the combination weights take the form of the posterior over the mixture model, making each kernel most influential in the region of input space for which it is most responsible for the input distribution.

## 3.1 Maximum Entropy Discrimination

The maximum entropy discrimination (MED) formalism was first introduced in Jaakkola, Meila, and Jebara [1999c] and was shown to be a flexible generalization

of support vector machines. MED produces a solution that is a distribution of parameter models $P(\Theta)$ rather than finding a single parameter setting $\Theta^*$. We start by assuming we are given training examples $X_t \in \Re^d$ with their corresponding labels $y_t \in \{\pm 1\}$ for $t \in \mathcal{T} = \{1, \ldots, T\}$.

We begin with a linear discriminant function, $\mathcal{L}(X; \Theta) = \theta^T X + b$. This discriminant function is specified by $\Theta = \{\theta, b\}$ containing the hyperplane parameter as well as a scalar bias value $b$. A linear classifier will make a binary prediction, $\hat{y}$, for an example, $X$, using the sign of discriminant.

$$\hat{y} = \text{sign}(\mathcal{L}(X; \Theta))$$

The *margin* of the classifier for example $X_t$ is given by $y_t \; \mathcal{L}(X_t; \Theta)$ and is positive iff the label $y_t$ is on the correct side of the hyperplane. We define the loss, $L : \mathbb{R} \to \mathbb{R}$, as a non-increasing and convex function of the margin.

If we were using a regularization theory framework [Girosi et al., 1995], we would seek a specific $\Theta^*$ setting of $\theta$ and $b$ that minimizes loss while ensuring that the magnitude of a regularization function $R(\Theta)$ is kept small for generalization purposes, e.g.,

$$\Theta^* = \underset{\Theta}{\text{argmin}} \left\{ R(\Theta) + \sum_{t \in \mathcal{T}} L\left(y_t \; \mathcal{L}(X_t; \Theta)\right) \right\}.$$

We can introduce desired minimum margin parameters, $\gamma = \{\gamma_1, \ldots, \gamma_T\}$ and rewrite the optimization of $\Theta^*$ as follows:

$$(\Theta^*, \gamma^*) = \min_{\Theta, \gamma} \left\{ R(\Theta) + \sum_{t \in \mathcal{T}} L(\gamma_t) \right\}$$

$$\text{subject to} \quad y_t \; \mathcal{L}(X_t; \Theta) - \gamma_t \; \geq \; 0, \quad \forall t \in \mathcal{T}.$$

In the case of MED, we instead seek a distribution over $\Theta$. Thus, classification will be performed with

$$\hat{y} = \text{sign}\left( \int_\Theta P(\Theta) \mathcal{L}(X; \Theta) d\Theta \right).$$

Again, we introduce margin parameters and express the classification constraints as

$$\int_{\Theta} P(\Theta)\,(y_t\ \mathcal{L}(X_t;\Theta))\,d\Theta\ \geq\ \gamma_t \quad \forall t \in \mathcal{T}.$$

Now, we must find $P(\Theta)$ that satisfies the classification constraints without assuming anything additional about $P(\Theta)$. This can be accomplished by maximizing the *Shannon Entropy*, $H(P(\Theta)) = -\int_{\Theta} P(\Theta)\log P(\Theta)d\Theta$, of the parameter distribution, while satisfying the constraints. This is a *maximum entropy* (ME) estimation problem. Though we retain the name "maximum entropy discrimination," we will actually use the more general minimum relative entropy (MRE), as in Jaakkola et al. [1999c]. We use the relative Shannon entropy given by

$$D(P\|P^{(0)}) = \int_{\Theta} P(\Theta)\ln\frac{P(\Theta)}{P^{(0)}(\Theta)}d\Theta,$$

which allows choice of a *prior* distribution, $P^{(0)}(\Theta)$. Note that minimizing relative entropy is more general since choosing $P^{(0)}(\Theta)$ uniform gives maximum entropy.

**Theorem 1** *[Jaakkola et al., 1999c] The solution to the MRE problem has the following general form:*

$$P(\Theta,\gamma) = \frac{1}{Z(\lambda)}P^{(0)}(\Theta,\gamma)e^{\sum_{t\in\mathcal{T}}\lambda_t[y_t\mathcal{L}(X_t|\Theta)-\gamma_t]}$$

*where $Z(\lambda)$ is the normalization constant (partition function) and $\lambda = \{\lambda_1,\ldots,\lambda_T\}$ defines a set of non-negative Lagrange multipliers, one for each classification constraint. $\lambda$ are set by finding the unique maximum of the following jointly concave function:*

$$J(\lambda) = -\log Z(\lambda).$$

We now have the means to solve MED problems, as long as we can analytically evaluate the partition function, $Z(\lambda)$. Because the objective, $J(\lambda)$ is concave, we can find its maximum through standard convex programming techniques [Boyd and Vandenberghe, 2003].

A primary advantage of the MED framework is that we can easily move away from the traditional linear discriminant to include probabilistic models. Log ratio discriminants of the form

$$\mathcal{L}(X;\Theta) = \ln \frac{P(X|\theta^+)}{P(X|\theta^-)} + b$$

are readily handled by MED. This discriminant function is specified by $\Theta = \{\theta^+, \theta^-, b\}$ containing generative model parameters as well as a scalar bias value $b$. Note that when $P(X|\theta^+) = \mathcal{N}(X|\mu^+, I)$ and $P(X|\theta^-) = \mathcal{N}(X|\mu^-, I)$ we have equal covariance Gaussians, and so retain a linear discriminant function.

Recall that in MED we solve for a distribution over solutions $P(\Theta)$ such that the *expected* value of the discriminant under this distribution agrees with the labeling. In addition to finding a $P(\Theta)$ that satisfies classification constraints in the expectation, MED regularizes the solution distribution $P(\Theta)$ by either maximizing its entropy or minimizing its relative entropy toward some prior target distribution $P^{(0)}(\Theta)$. Thus, MED solves the constrained optimization problem

$$\min_{P(\Theta)} D(P\|P^{(0)}) \text{ s.t. } \int_\Theta P(\Theta)[y_t\mathcal{L}(X_t;\Theta) - \gamma_t] \geq 0 \quad \forall t \in \mathcal{T} \tag{3.1}$$

which projects the prior $P^{(0)}(\Theta)$ to the closest point in the admissible set or convex hull defined by the above $t = 1, \ldots, T$ constraints. The solution[Jaakkola et al., 1999c] for the posterior $P(\Theta)$ is the standard maximum entropy setting

$$P(\Theta) = \frac{P^{(0)}(\Theta)}{Z(\lambda)} e^{\sum_t \lambda_t [y_t \mathcal{L}(X_t;\Theta) - \gamma_t]}$$
$$\text{where } Z(\lambda) = \int_\Theta P^{(0)}(\Theta) e^{\sum_t \lambda_t [y_t \mathcal{L}(X_t;\Theta) - \gamma_t]}.$$

The partition function $Z(\lambda)$ normalizes $P(\Theta)$. MED finds the optimal setting of the Lagrange multipliers $\lambda_t$ for $t = 1, \ldots, T$ by maximizing the concave objective function $J(\lambda) = -\ln Z(\lambda)$. Given $\lambda$, the solution distribution $P(\Theta)$ is fully specified. It is then straightforward to use this distribution for predicting the label of a new datum $X$ via $\hat{y} = \text{sign}(\int_\Theta P(\Theta)\mathcal{L}(X;\Theta)d\Theta)$. The integrals in MED are often analytic and result in

efficient and deterministic equations for both learning and prediction. Exponential family models were shown to be analytic in Jebara [2004a].

## 3.2 Discriminative Gaussian Ratio Classifiers

We now derive a log-likelihood ratio Gaussian classifier. We want to find a discriminant function as follows:

$$\mathcal{L}(X;\Theta) = \ln \frac{\mathcal{N}(X|\mu^+, I)}{\mathcal{N}(X|\mu^-, I)} + b$$

whose sign agrees with the labeled training set. This discriminant function is specified by $\Theta = \{\mu^+, \mu^-, b\}$ containing both Gaussian means $\mu^+$ and $\mu^-$ as well as a scalar bias value $b$. Thus, MED solves the constrained optimization problem (3.1).

Interestingly, applying MED to a ratio of identity covariance Gaussians *exactly reproduces* support vector machines, as is evident from the objective, $J_{\text{SVM}}$. We assume that the prior distribution factorizes into a prior over the vector parameters and a prior over scalar bias, $P^{(0)}(\Theta) = P^{(0)}(\mu^+)P^{(0)}(\mu^-)P^{(0)}(b)$. The first two priors are identity-covariance zero-mean Gaussians over the means as follows $P^{(0)}(\mu^+) = \mathcal{N}(\mu^+|0, I)$ and $P^{(0)}(\mu^-) = \mathcal{N}(\mu^-|0, I)$ which encourages means of low magnitude for our Gaussians. The last prior is a non-informative (i.e. flat) prior $P^{(0)}(b) = \mathcal{N}(b|0, \infty)$ indicating that any scalar bias is equally probable *a priori*. The integrals are solved in Jebara [2004a]. The resulting objective function is

$$J_{\text{SVM}}(\lambda) = \sum_{t\in\mathcal{T}} \lambda_t\gamma_t - 2 \times \frac{1}{2} \sum_{t,t'\in\mathcal{T}} \lambda_t\lambda_{t'}y_ty_{t'}X_t^TX_{t'}.$$

Note the quadratic terms appear doubled since we have a term for each of the two Gaussian models in the partition function. If we choose $\gamma_t = 2$ the above optimization is identical to the SVM dual optimization problem (it is simply twice the SVM's objective). Our objective is also subject to the standard non-negativity constraints on the $\lambda_t$ values since the expectation constraints in the maximum entropy problem

use *greater-than* inequalities. Furthermore, the non-informative bias prior yields the equality constraint $\sum_t \lambda_t y_t = 0$. Thus learning involves simply solving a quadratic program (QP). Prediction for a query datum $X$ is given by

$$\int_\Theta P(\Theta)\mathcal{L}(X;\Theta)d\Theta \;=\; \sum_{t\in\mathcal{T}} y_t\lambda_t X_t^T X + \hat{b},$$

where the expected bias $\hat{b}$ is found via the Karush-Kuhn-Tucker (KKT) conditions as usual.

To work with non-separable problems, we use a distribution over margins in the prior and posterior instead of simply setting margins equal to a constant 2 (which is like using a delta-function prior $P^{(0)}(\gamma_t) = \delta(\gamma_t - 2)$). The MED solution distribution then involves an augmented $\Theta$ which includes all margin variables as follows: $\Theta = \{\mu^+, \mu^-, b, \gamma_1, \ldots, \gamma_T\}$. The formula for the partition function $Z(\lambda)$ is as above except we now have $P^{(0)}(\Theta) = P^{(0)}(\mu^+)P^{(0)}(\mu^-)P^{(0)}(b)\prod_{t=1}^T P^{(0)}(\gamma_t)$ and integrate over $d\Theta d\gamma_1 \ldots d\gamma_T$. The margin priors are chosen to favor large margins yet allow negative margins with exponentially decaying probability. An appropriate choice is

$$P^{(0)}(\gamma_t) = \begin{cases} ce^{-c(2-\gamma_t)} & \gamma_t \leq 2, \\ P^{(0)}(\gamma_t) = 0 & \text{otherwise.} \end{cases}$$

Solving gives the new objective function $J(\lambda) = J_{SVM}(\lambda) + \sum_t \ln(1 - \lambda_t/c)$. The non-separability creates a logarithmic barrier function preventing Lagrange multipliers from growing beyond $c$. This is like a soft margin SVM's explicit constraint $\lambda_t \leq c$. For simplicity, we omit these logarithmic barrier functions in all non-separable MED problems, and just impose the upper bound $c$ constraint on Lagrange multipliers.

Another important extension to the above is allowing the Gaussian models to be applied over feature vectors instead of the original data. In this situation, we have:

$$\mathcal{L}(X;\Theta) = \ln\frac{\mathcal{N}(\phi^+(X)|\mu^+, I)}{\mathcal{N}(\phi^-(X)|\mu^-, I)} + b \;=\; \ln\frac{\exp(-\frac{1}{2}\|\phi^+(X) - \mu^+\|^2)}{\exp(-\frac{1}{2}\|\phi^-(X) - \mu^-\|^2)} + b$$

where the vector $X$ is mapped into a feature vector $\phi^+(X)$ before interacting with the positive Gaussian model and a feature vector $\phi^-(X)$ before interacting with the

negative Gaussian model. Because we are dealing with possibly infinite dimensional feature spaces, the Gaussian means are handled as linear combinations of distances between examples implicitly projected to feature space via kernel functions. Equations for Hilbert space Gaussians with implicit mean are easily derived by starting with traditional distance and using the well-known $d(x,y) = k(x,x) + k(y,y) - 2k(x,y)$. Since we are dealing with discrimination, we can ignore normalization issues with these Gaussian models. The resulting $J(\lambda)$ objective, subject to SVM-like constraints, is still a QP:

$$2\sum_{t\in\mathcal{T}} \lambda_t - \frac{1}{2}\sum_{t,t'\in\mathcal{T}} \lambda_t\lambda_{t'}y_t y_{t'}(k^+(X_t,X_{t'}) + k^-(X_t,X_{t'}))$$

where Mercer kernel functions $k^+(X_t,X_{t'}) = \phi^+(X_t)^T\phi^+(X_{t'})$ and $k^-(X_t,X_{t'}) = \phi^-(X_t)^T\phi^-(X_{t'})$ take the place of inner products of our feature functions. The prediction rule for the final classifier for a query datum $X$ is $\hat{y} = \frac{1}{2}\sum_t y_t\lambda_t(k^+(X,X_t) + k^-(X,X_t)) + \hat{b}$.

## 3.3 Latent Maximum Entropy Discrimination

We now present the latent MED model for exponential family distributions. The discriminant function is written as follows:

$$\mathcal{L}(X_t;\Theta) = \ln\frac{\sum_{m=1}^{M} P(m,\phi_m^+(X_t)|\theta_m^+)}{\sum_{n=1}^{N} P(n,\phi_n^-(X_t)|\theta_n^-)} + b. \tag{3.2}$$

In Equation (3.2), we introduce model parameters $\theta_m^+$, $\theta_n^-$, for positive and negative models, respectively; and $b$ for the scalar bias. A non-informative prior is chosen for $b$ to formalize our lack of knowledge about the parameter and to simplify the resulting MED integrals. Appropriate priors should be selected for $\theta_m^+$ and $\theta_n^-$. We wish to recover an MED distribution,

$$P(\Theta) = P(\theta_1^+,\ldots,\theta_M^+,\theta_1^-,\ldots,\theta_N^-,b),$$

that is as close to these priors as possible yet also satisfies the classification constraints which should label the data correctly under expectations over $P(\Theta)$. Note the use

of explicit feature mappings $\phi_m^+$, $\phi_n^-$ which permit each model to reside in a distinct feature space.

In the latent variable case the classical maximum entropy solution (3.1) is not fruitful because, for mixtures, computing and minimizing the partition function $Z(\lambda)$ is intractable with exponentially many terms. This mirrors the classical case of maximum likelihood parameter estimation for mixture models, for which we use expectation maximization (EM) [Dempster et al., 1977]. To compensate, we use variational methods. The general scheme is to use tractable functions that lower and upper bound the intractable terms of the partition function. We then optimize the tractable functions to indirectly optimize the intractable one, subject to tightness of the bounds. We iterate the bounding projection and re-estimation of the parameters, thus tightening the bound and maximizing the objective, until convergence. The procedure is guaranteed to converge to a local maximum as in Jaakkola et al. [1999c].

Jensen's inequality is first applied to the correct class model in the classification constraints to simplify them, while tightening them so that any admissible point is admissible under the original constraints. Then, Jensen is applied to the partition function which still contains log-sums for the incorrect class model. This yields a tractable latent MED projection. Starting with an initial solution $P^{(1)}(\Theta)$, we iterate the tractable projection step and convex hull restriction step until convergence, slowly updating the current MED solution $P^{(i)}(\Theta)$ and reducing divergence to the prior $P^{(0)}(\Theta)$, while moving within the MED admissible set.

We now begin to derive the tractable partition. We illustrate the variational lower bound by considering the classification constraint for the $t^{\text{th}}$ example, which we assume without loss of generality has a positive label, $y_t = 1$. The constraint for the $t^{\text{th}}$ datum is

$$\int P(\Theta) \left( \ln \frac{\sum_m P(m, \phi_m^+(X_t)|\theta_m^+)}{\sum_n P(n, \phi_n^-(X_t)|\theta_n^-)} + b - \gamma_t \right) d\Theta \geq 0.$$

Choosing a distribution $q_t$, we apply Jensen's inequality to the positive mixture model

to obtain a *stricter* classification constraint:

$$\int P(\Theta)\Big(\sum_m q_t(m)\ln P(m,\phi_m^+(X_t)|\theta_m^+) + H(q_t) \\ -\ln\sum_n P(n,\phi_n^-(X_t)|\theta_n^-) + b - \gamma_t\Big)d\Theta \geq 0$$

where $H(q_t)$ is the entropy of $q_t$. We choose a distribution $q_t$ such that the classification inequality is tight under $P^{(i-1)}(\Theta)$. We see that the expression for the bounding constraint is simpler, but still includes a log-sum for the negative mixture model. The remaining log-sum will be eliminated by applying Jensen's inequality to upper bound the partition function, as we will show.

We now proceed to apply Jensen's inequality on each constraint's numerator for positive data $t \in \mathcal{T}^+$ and on each constraint's denominator for negative data $t \in \mathcal{T}^-$. This leads to a new, more constrained problem in which we formally define a single MED projection as follows:

$$\min_{P(\Theta)} D(P(\Theta)\|P^{(0)}(\Theta))$$

subject to:

$$E_P\left\{\sum_m q_t(m)\ln P(m,\phi_m^+(X_t)|\theta_m^+) + H(q_t) - \ln\sum_n P(n,\phi_n^-(X_t)|\theta_n^-) + b\right\} \geq \gamma_t \forall t \in \mathcal{T}^+$$

$$E_P\left\{\sum_n q_t(n)\ln P(n,\phi_n^-(X_t)|\theta_n^-) + H(q_t) - \ln\sum_m P(m,\phi_m^+(X_t)|\theta_m^+) - b\right\} \geq \gamma_t \forall t \in \mathcal{T}^-.$$

Thus, the Jensen-bounded correct mixture dominates the incorrect mixture on all data points. The new partition function for this projection given $q$ is

$$\dot{Z}(\lambda|q) = \int_\Theta P^{(0)}(\Theta)\exp\Big(\sum_{t\in\mathcal{T}^+}\lambda_t(\sum_m q_t(m)\ln P(m,\phi_m^+(X_t)|\theta_m^+) + H(q_t) \\ -\ln\sum_n P(n,\phi_n^-(X_t)|\theta_n^-) + b - \gamma_t)\Big) \\ \exp\Big(\sum_{t\in\mathcal{T}^-}\lambda_t(\sum_n q_t(n)\ln P(n,\phi_n^-(X_t)|\theta_n^-) + H(q_t) \\ -\ln\sum_m P(m,\phi_m^+(X_t)|\theta_m^+) - b - \gamma_t)\Big)d\Theta.$$

It is interesting to note that this partition function bounds the original one via $Z(\lambda) \geq \dot{Z}(\lambda|q)$ for any $q$. To project, we maximize:

$$\lambda^* = \underset{\lambda}{\text{argmax}}(-\ln \dot{Z}(\lambda|q))$$

$$\text{subject to:}$$

$$0 \leq \lambda_t \leq c, \quad \sum_{t \in \mathcal{T}} \lambda_t y_t = 0.$$

This problem is log-concave and simpler than the original full MED problem, yet it is still intractable due to the remaining log-sums in the expression from the opposite class models. Therefore, we place an upper bound on the partition function for the constrained projection by applying Jensen's inequality to the log-sums in $\dot{Z}(\lambda|q)$ using variational distributions $Q_t$. This gives a variational upper bound $\ddot{Z}(\lambda, Q|q) \geq \dot{Z}(\lambda|q)$:

$$\begin{aligned}\ddot{Z}(\lambda, Q|q) = \int_\Theta P^{(0)}(\Theta) \exp\Big( \sum_{t \in \mathcal{T}^+} \lambda_t (\sum_m q_t(m) \ln P(m, \phi_m^+(X_t)|\theta_m^+) + H(q_t) \\ - \sum_n Q_t(n) \ln P(n, \phi_n^-(X_t)|\theta_n^-) - H(Q_t) + b - \gamma_t)\Big) \\ \exp\Big( \sum_{t \in \mathcal{T}^-} \lambda_t (\sum_n q_t(n) \ln P(n, \phi_n^-(X_t)|\theta_n^-) + H(q_t) \\ - \sum_m Q_t(m) \ln P(m, \phi_m^+(X_t)|\theta_m^+) - H(Q_t) - b - \gamma_t)\Big) d\Theta.\end{aligned}$$

To tighten this upper bound, we must minimize the partition function $\ddot{Z}$ over $Q$ or maximize $-\ln \ddot{Z}$ over $Q$. Thus, we have:

$$(\lambda^*, Q^*) = \underset{\lambda, Q}{\text{argmax}}(-\ln \ddot{Z}(\lambda, Q|q))$$

$$\text{subject to:}$$

$$0 \leq \lambda_t \leq c, \quad \sum_{t \in \mathcal{T}} \lambda_t y_t = 0$$

$$0 \leq Q, \quad Q_t^T 1 = 1$$

The above maximization uses a *tractable* partition function $\ddot{Z}(\lambda, Q|q)$ which includes the variational distributions $Q$ and $q$ required by the double Jensen bound. In

fact, we can solve the optimization *simultaneously* over $\lambda$ and $Q$ as a convex program. The q distribution is updated as the maximum likelihood posterior over the latent variables, given $Q$ and $\lambda$. The overall latent MED problem proceeds iteratively by updating the $q$ to tighten the equality constraints around a previous $P^{(i-1)}(\Theta)$ model to get a more constrained projection and then solves the constrained projection as a convex program, recovering $Q$ and $\lambda$ simultaneously. This iterative procedure—bounding the classification constraints with Jensen using a previous model $P^{(i-1)}(\Theta)$ and then solving a simpler MED problem—has been shown to converge to a local minimum as in the latent anomaly detection problem of Jaakkola et al. [1999c].

The derivation we present here is an improvement upon the presentation in Jebara [2004a] because of the application of Jensen's inequality to bound the incorrect-class model. Our double Jensen approach yields a tight upper bound rather than the *winner takes all* approximation. Our new derivation also makes it apparent that latent MED parameter estimation is a minimax optimization. The two opposing bounds must be tight, squeezing in on the true latent MED objective:

$$\operatorname*{argmin}_{Q} \operatorname*{argmax}_{q} \ddot{Z}(\lambda, Q|q) = Z(\lambda).$$

## 3.4 Discriminative Kernelized Gaussian Mixtures

We now present the latent MED model for the ratio of Hilbert space Gaussian mixtures. The discriminant function is written as follows:

$$\mathcal{L}(X_t;\Theta) = \ln \frac{\sum_{m=1}^{M} \alpha_m \mathcal{N}(\phi_m^+(X_t)|\mu_m^+, I)}{\sum_{n=1}^{N} \beta_n \mathcal{N}(\phi_n^-(X_t)|\mu_n^-, I)} + b \tag{3.3}$$

and the MED classifier is:

$$\hat{y} = \text{sign}\left(\int_{\Theta} P(\Theta)\mathcal{L}(X_t;\Theta)d\Theta\right). \tag{3.4}$$

In Equation (3.3), we introduce model parameters $\mu_m^+$, $\mu_n^-$, for Gaussian means; $\alpha$, $\beta$ for mixing proportions; and $b$ for the scalar bias. White Gaussian prior distributions,

$P^{(0)}(\mu) = \mathcal{N}(0, I)$, are chosen to regularize the Gaussian means toward zero. Non-informative priors are chosen for $\alpha$, $\beta$, and $b$ to express our lack of knowledge about these parameters and to simplify the resulting MED integrals. We wish to recover an MED distribution that is as close to these priors as possible yet also satisfies the classification constraints which should label the data correctly under expectations over $P(\Theta)$. Note the use of explicit feature mappings $\phi_m^+$, $\phi_n^-$ which permit each Gaussian to reside in a distinct feature space.

Specifically, we wish to recover

$$P(\Theta) = P(\alpha_1, \mu_1^+, \ldots, \alpha_M, \mu_M^+, \beta_1, \mu_1^-, \ldots, \beta_N, \mu_N^-, b).$$

We apply the variational approach developed in Section 3.3 with the conditioned Gaussian models replacing the generic joint probabilities. The more substantial task is to solve the various integrals for $\ddot{Z}(\lambda, Q|q)$ in closed form. We refer to the formulas in Jebara [2004a, chap. 5]. The expression for the partition function drastically simplifies if we assume a non-informative prior on the bias and non-informative Dirichlet priors over the multinomial parameters. Ultimately, the negative logarithm of the partition function is computed and is our SVM-like objective function $\ddot{J}(\lambda, Q|q) = -\log \ddot{Z}(\lambda, Q|q)$.

$$\begin{aligned}
\ddot{J}(\lambda, Q|q) \quad = \quad & \sum_{t \in \mathcal{T}} \lambda_t (H(Q_t) - H(q_t)) + \sum_{t \in \mathcal{T}} \lambda_t \gamma_t \\
& - \frac{1}{2} \sum_{\substack{t \in \mathcal{T}^+ \\ t' \in \mathcal{T}^+}} \lambda_t \lambda_{t'} \Big( \sum_m q_t(m) q_{t'}(m) k_m^+(t, t') + \sum_n Q_t(n) Q_{t'}(n) k_n^-(t, t') \Big) \\
& - \frac{1}{2} \sum_{\substack{t \in \mathcal{T}^- \\ t' \in \mathcal{T}^-}} \lambda_t \lambda_{t'} \Big( \sum_m Q_t(m) Q_{t'}(m) k_m^+(t, t') + \sum_n q_t(n) q_{t'}(n) k_n^-(t, t') \Big) \\
& + \sum_{\substack{t \in \mathcal{T}^+ \\ t' \in \mathcal{T}^-}} \lambda_t \lambda_{t'} \Big( \sum_m q_t(m) Q_{t'}(m) k_m^+(t, t') + \sum_n Q_t(n) q_{t'}(n) k_n^-(t, t') \Big).
\end{aligned}$$

Above, we have written the objective in terms of kernel evaluations, $k_m^+(\cdot, \cdot)$ and

$k_n^-(\cdot,\cdot)$, rather than inner products on our explicit feature mappings.

A set of $M+N$ linear equality constraints emerge from barrier functions during integration and are enforced in addition to the positivity and upper bound constraints on $\lambda$. Note that these constraints subsume the traditional linear equality constraint of the SVM.

$$\sum_{t\in\mathcal{T}^-} \lambda_t Q_t(m) = \sum_{t\in\mathcal{T}^+} \lambda_t q_t(m) \;\; \forall m \in \{1,\ldots,M\}$$
$$\sum_{t\in\mathcal{T}^+} \lambda_t Q_t(n) = \sum_{t\in\mathcal{T}^-} \lambda_t q_t(n) \;\; \forall n \in \{1,\ldots,N\}$$
$$0 \le \lambda_t \le c \;\; \forall t \in \{1,\ldots,T\}$$

We implemented the maximization of $\ddot{J}(\lambda, Q|q)$, subject to the above constraints, as a quadratic program, omitting the entropy terms $H(Q_t)$ due to their non-quadratic nature. This simplification is tolerable for the moment, because $H(Q_t)$ vanishes toward zero as $Q$ becomes committed to a single mixture component, which often happens in practice.

The update for $q$ ensures that bounds on classification constraints are tight under the previous iteration's $P^{(i-1)}(\Theta)$ solution:

$$q_t(m) \;\propto\; \exp\left(E\{\ln\alpha_m\} + E\{\ln\mathcal{N}(\phi_m^+(X_t)|\mu_m^+)\}\right) \;\; t\in\mathcal{T}^+$$
$$q_t(n) \;\propto\; \exp\left(E\{\ln\beta_n\} + E\{\ln\mathcal{N}(\phi_n^-(X_t)|\mu_n^-)\}\right) \;\; t\in\mathcal{T}^-.$$

Thus, for updating $q$ and for making predictions with the final model, we need expectations under $P(\Theta)$ for various components of the discriminant function. For

instance, the expected value of each log-Gaussian for $m = 1..M$ is

$$\begin{aligned} E\{\ln \mathcal{N}(\phi_m^+(X_t)|\mu_m^+)\} &= -\frac{D}{2}\ln(2\pi) - \frac{1}{2} - \frac{1}{2}k_m^+(X_t, X_t) \\ &+ \sum_{\tau\in\mathcal{T}^+} \lambda_\tau q_\tau(m) k_m^+(X_\tau, X_t) - \sum_{\tau\in\mathcal{T}^-} \lambda_\tau Q_\tau(m) k_m^+(X_\tau, X_t) \\ &- \frac{1}{2}\sum_{\substack{\tau\in\mathcal{T}^+ \\ \tau'\in\mathcal{T}^+}} \lambda_\tau\lambda_{\tau'} q_\tau(m) q_{\tau'}(m) k_m^+(X_\tau, X_{\tau'}) \\ &- \frac{1}{2}\sum_{\substack{\tau\in\mathcal{T}^- \\ \tau'\in\mathcal{T}^-}} \lambda_\tau\lambda_{\tau'} Q_\tau(m) Q_{\tau'}(m) k_m^+(X_\tau, X_{\tau'}) \\ &+ \sum_{\substack{\tau\in\mathcal{T}^+ \\ \tau'\in\mathcal{T}^-}} \lambda_\tau\lambda_{\tau'} q_\tau(m) Q_{\tau'}(m) k_m^+(X_\tau, X_{\tau'}). \end{aligned}$$

We similarly compute the expected log Gaussian probability $E\{\ln \mathcal{N}(\phi_n^-(X_t)|\mu_n^-)\}$ for negative models. Finally, we obtain the expected bias and mixing proportions indirectly via the following surrogate variables:

$$\begin{aligned} a_m &= E\{\ln \alpha_m\} + \frac{1}{2}E\{b\} \quad \forall m \in \{1, \ldots, M\} \\ b_n &= E\{\ln \beta_n\} - \frac{1}{2}E\{b\} \quad \forall n \in \{1, \ldots, N\} \end{aligned}$$

This is done by using the KKT conditions, which ensure that at non-bound Lagrange multiplier settings, classification inequalities are achieved *with equality*. In other words, the stricter MED constraints implicit in $\ddot{Z}(\lambda, Q|q)$ become equalities. Thus,

when $\lambda_t \in (0, c)$ we must achieve the following with equality:

$$\begin{aligned}
\sum_m q_t(m)(a_m + E\{\ln \mathcal{N}(\phi_m^+(X_t)|\mu_m^+)\}) + H(q_t) \quad &= \\
\sum_n Q_t(n)(b_n + E\{\ln \mathcal{N}(\phi_n^-(X_t)|\mu_n^-)\}) + H(Q_t) + \gamma_t \quad &t \in \mathcal{T}^+ \\
\sum_n q_t(n)(b_n + E\{\ln \mathcal{N}(\phi_n^-(X_t)|\mu_n^-)\}) + H(q_t) \quad &= \\
\sum_m Q_t(m)(a_m + E\{\ln \mathcal{N}(\phi_m^+(X_t)|\mu_m^+)\}+) H(Q_t) + \gamma_t \quad &t \in \mathcal{T}^-.
\end{aligned}$$

We solve for $a_m$ for $m = 1..M$ and $b_n$ for $n = 1..N$ in this (over-constrained) linear system, obtaining the expected bias and mixing proportions.

To make predictions with the latent MED's learned $P(\Theta)$, we cannot use the standard rule from Equation (3.4) due to its intractable log-sums. Instead, we employ the following approximate prediction using the expectation formulas we just derived:

$$\hat{y} = \ln \frac{\sum_m \exp\left(E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+)\} + a_m\right)}{\sum_n \exp\left(E\{\ln \mathcal{N}(\phi_n^-(X)|\mu_n^-)\} + b_n\right)}. \tag{3.5}$$

## 3.5 Sequential Minimal Optimization

Platt's sequential minimal optimization (SMO) [Platt, 1999] is an algorithm for the iterative solution of convex quadratic programs with a single linear equality constraint and bound constraints. SMO was specifically designed to solve large-scale SVM optimization problems. SMO is particularly efficient when the solution is sparse, as is the case with SVM.

SMO takes advantage of a special case of the structured constraints we describe in the next section. SMO can be applied to a QP problem

$$x^* = \operatorname*{argmin}_x \left(c^T x + \frac{1}{2} x^T \mathbf{H} x\right), \quad \text{subject to } \mathbf{A}x = b. \tag{3.6}$$

The single linear equality constraint has the form:

$$\begin{bmatrix} \dots & a_u & \dots & a_v & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ x_u \\ \vdots \\ x_v \\ \vdots \end{bmatrix} = b. \tag{3.7}$$

If we examine the form of the linear equality constraint (3.7), we notice that we can maintain the constraint by updating as few as two arbitrary dimensions, $u$ and $v$. This is accomplished by maintaining the equality,

$$a_u \hat{x_u} + a_v \hat{x_v} = a_u x_u + a_v x_v \tag{3.8}$$

where $\hat{x}$ denotes the new value of $x$. We can solve (3.8) for $x_u$ and rewrite the quadratic objective function (3.6) in terms of the two axes $u$ and $v$ in the one variable $x_u$. The resulting function can be differentiated with respect to $x_u$, and the root can be computed to find the minimum. Iterating such a procedure for all pairs of axes will monotonically approach the global minimum of (3.6) while maintaining the constraint. See [Platt, 1999] for details.

SMO relies on several heuristics to choose axis-pairs to optimize. These heuristics are critical to the efficiency of SMO. The algorithm takes maximum advantage of sparsity and attempts to make maximum progress with each step. Building upon Platt's work, others have proposed improved heuristics for axis selection [Keerthi et al., 1999].

We now consider the problem of optimizing the latent MED quadratic program subject to the given system of linear equality constraints. Like many optimization problems from the machine learning community, this problem possesses structure that can be exploited to yield an efficient algorithm. The general form of the objective function for the MED discriminative projection step is $J(\lambda) = c^T\lambda + \frac{1}{2}\lambda^T\mathbf{H}\lambda$, where $\lambda$ is the solution to the QP proposed in the Section 3.4. The constraints are of the

form $\mathbf{A}\lambda = 0$. In general, we cannot simultaneously maintain the $M + N$ linear equality constraints by optimizing over only axis pairs, as in Platt's SMO for the SVM. However, some analysis reveals that we can reduce the number of axes to a maximum of $\max(M, N)$ and a minimum of two, depending on the situation.

General linear equality constrained algorithms must consider the null space of the linear system of constraints. In the case of constrained Newton methods, this generally requires the inversion of a matrix that corresponds to the KKT system of the problem. Working set methods, an extreme case of which is Platt's SMO, save computation by optimizing over a subset of the axes in an iterative algorithm. SMO uses a working set of two axes. The single linear equality constraint of the SVM can be explicitly satisfied between the two axes. There is one degree of freedom along which the linearly coupled variables are analytically optimized. The primary efficiency of SMO is that it takes advantage of sparsity in $\lambda$.

Latent MED is more general than the SVM. We replicate variables to permit optimization over a latent distribution. This replication imparts regular structure to the constraints. The following equation shows the form of the constraint matrix $\mathbf{A}$ for a latent MED problem with four latent states, two for each binary class from which examples are drawn:

$$\begin{bmatrix} \dots & q_{u_1} & q_{u_1} & \dots & -1 & 0 & \dots & q_{w_1} & q_{w_1} & \dots \\ \dots & q_{u_2} & q_{u_2} & \dots & 0 & -1 & \dots & q_{w_2} & q_{w_2} & \dots \\ \dots & 1 & 0 & \dots & -q_{v_1} & -q_{v_1} & \dots & 1 & 0 & \dots \\ \dots & 0 & 1 & \dots & -q_{v_2} & -q_{v_2} & \dots & 0 & 1 & \dots \end{bmatrix} \begin{bmatrix} \vdots \\ \lambda_{u_1} \\ \lambda_{u_2} \\ \vdots \\ \lambda_{v_1} \\ \lambda_{v_2} \\ \vdots \\ \lambda_{w_1} \\ \lambda_{w_2} \\ \vdots \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{3.9}$$

The illustrated columns of $\mathbf{A}$ correspond to axes $u_1, u_2$, $v_1, v_2$, and $w_1, w_2$ associated with examples $u$, $v$, and $w$ in the latent MED problem. The entries of the form $q_{i_j}$ are given by the posterior probability distributions $q_i$ over the $M$ and $N$ latent states of the generative models. (In this case, $M = N = 2$.) Note the replication of values within the constraint matrix. A corresponding replication is required in the dual variables, $\lambda$. The embedded $N \times N$ and $M \times M$ identity matrices effectively select a latent configuration of the *opposite* class and its associated dual variable. Note how the form of the constraints differs based upon the class of the chosen example. We will now explore the effect of this structure on the optimization problem.

### 3.5.1 Inter-class

Let us consider two examples, $u$ and $v$ from opposite classes. Without loss of generality, let $u$ be positive and $v$ be negative. Then we can maintain the constraints using the following equalities in vector form:

$$q_u\big(\hat{\lambda_u}^T 1\big) - \hat{\lambda_v} = q_u\big(\lambda_u^T 1\big) - \lambda_v$$
$$\hat{\lambda_u} - q_v\big(\hat{\lambda_v}^T 1\big) = \lambda_u - q_v\big(\lambda_v^T 1\big).$$

Then, we can write

$$\Delta\lambda_v = (\Delta\lambda_u^T 1)\ q_u$$
$$\Delta\lambda_u = (\Delta\lambda_v^T 1) q_v = (((\Delta\lambda_u^T 1) q_u)^T 1) q_v = (\Delta\lambda_u^T 1)\ q_v$$

Thus, $(\Delta\lambda_u^T 1) = (\Delta\lambda_v^T 1)$, and $\Delta\lambda_u$ and $\Delta\lambda_v$ are coupled via their norms by a single scalar that we will call $\Delta s$.

$$\Delta\lambda_v = \Delta s\ q_u \tag{3.10}$$
$$\Delta\lambda_u = \Delta s\ q_v. \tag{3.11}$$

In other words, we can maintain the linear equality constraints by updating the solution with the scaled posterior distributions $q_u$ and $q_v$.

The change in the quadratic objective function for the axes $u$ and $v$ is

$$\begin{aligned}\Delta J_{uv}(\Delta\lambda) = c_u^T\Delta\lambda_u + c_v^T\Delta\lambda_v \\ + \tfrac{1}{2}\Delta\lambda_u^T\mathbf{H}_{uu}\Delta\lambda_u + \Delta\lambda_u^T\mathbf{H}_{uv}\Delta\lambda_v + \tfrac{1}{2}\Delta\lambda_v^T\mathbf{H}_{vv}\Delta\lambda_v \\ + \sum\nolimits_{t\neq u,v}(\Delta\lambda_t^T\mathbf{H}_{tu}\Delta\lambda_u + \Delta\lambda_t^T\mathbf{H}_{tv}\Delta\lambda_v).\end{aligned}$$

All that remains is to express the change in the objective, $\Delta J_{uv}(\Delta\lambda)$ as a function of $\Delta s$ by changing variables using Equations (3.10) and (3.11). This is a straightforward algebraic manipulation. The resulting one-dimensional quadratic objective function, $\Delta J_{uv}(\Delta s)$, can be analytically optimized by finding the root of the derivative under the box constraint.

### 3.5.2 Intra-class

Now we examine the case in which axes $u$ and $w$ are chosen from the same class. Without loss of generality, we choose $u$ and $w$ members of the positive class as in Equation (3.9). As in the inter-class case, we write the constraints in terms of our two variables, $u$ and $w$, in vector form:

$$\begin{aligned}q_u\big(\hat{\lambda_u}^T 1\big) + q_w\big(\hat{\lambda_w}^T 1\big) &= q_u\big(\lambda_u^T 1\big) + q_w\big(\lambda_w^T 1\big) \\ \hat{\lambda_u} + \hat{\lambda_w} &= \lambda_u + \lambda_w\end{aligned}$$

These constraints are simpler than in the previous case. In particular, when $q_u = q_w$, we obtain the further simplification:

$$\big(\hat{\lambda_u}^T 1\big) + \big(\hat{\lambda_w}^T 1\big) = \big(\lambda_u^T 1\big) + \big(\lambda_w^T 1\big) \tag{3.12a}$$

$$\hat{\lambda_u} + \hat{\lambda_w} = \lambda_u + \lambda_w. \tag{3.12b}$$

Note that (3.12a) is satisfied when (3.12b) is satisfied. Therefore, we need only enforce that the sum of the vectors is constant. It is important to observe that the intra-class case is comprised of two sub-cases distinguished by whether $q_u = q_w$ or not.

**Intra-class equal** Now, we continue with the derivation of the *intra-class equal* case. Referring to Equation (3.12b), we see that the constraints can be satisfied, at each iteration, by choosing a single sub-axis $k$ along which to optimize, so we restrict ourselves to $u_k$ and $w_k$. In this case, the latent MED optimization decomposes directly into Platt's SMO. Working from (3.12b), we obtain

$$\hat{\lambda_{u_k}} + \hat{\lambda_{w_k}} = \lambda_{u_k} + \lambda_{w_k},$$

which leads us to

$$\begin{aligned}\hat{\lambda_{u_k}} &= \lambda_{u_k} + (\lambda_{w_k} - \hat{\lambda_{w_k}}) \\ \hat{\lambda_{w_k}} &= \lambda_{w_k} + (\lambda_{u_k} - \hat{\lambda_{u_k}}).\end{aligned}$$

We also observe that

$$\Delta\lambda_{u_k} = -\Delta\lambda_{w_k}.$$

We again call this scalar quantity $\Delta s$. The derivation of update rules proceeds analogously with Platt's SMO.

**Intra-class unequal** Finally, we consider the *intra-class unequal* case. This situation is slightly complicated by the fact that the posterior distributions $q_u$ and $q_w$ are not equal, and so no longer factor out and cancel as in (3.12a). This change requires us to explicitly enforce a constraint on the norm of the dual variable vectors, $\hat{\lambda_u}$ and $\hat{\lambda_w}$. In order to enforce the additional constraint, we must choose an additional sub-axis $l$ over which to optimize.

Ultimately, we express the relationship between the four changing axes as a single scalar $\Delta s$ and derive update rules, similarly to the other cases.

### 3.5.3 Newton Step

After implementing SMO as described above, we had trouble getting consistent convergence with latent variable problems. SMO did converge well in the SVM case. It seemed that convergence was heavily dependent upon the order in which axes were selected, which should not be the case for a convex optimization. After verifying the implementation, we were forced to accept that convergence could proceed with infinitesimal step size due to an almost flat objective when considering a minimal working set. It appears that F. R. Bach [2004] ran into a similar problem. To address this situation, we apply a more global optimization over a larger working set of axes. As long as the working set is a small percentage of the full set of axes, the computational efficiency is not harmed.

We use an equality constrained second-order gradient descent type step (a Newton step) as described in Boyd and Vandenberghe [2003]. This optimization requires that we construct and invert the KKT system to solve for the minimum of the objective within the null space of the equality constraints. We use a singular value decomposition (SVD) to perform the matrix pseudo-inverse. This is a reasonably efficient cubic time algorithm, but because the working set tends to be small, the time is still $O(n^2)$ on the full problem. This is in line with the average complexity of SMO.

### 3.5.4 Implementation

As with any piece of non-trivial software, we chose to begin with a relatively simple but correct implementation. We actually prototyped the SMO optimization function in MATLAB. We gave our SMO function an interface that was compatible with the interface for the MATLAB optimization toolbox implementation of QP (`quadprog`). Because MATLAB is an interactive language and built-in matrix operations, implementing and debugging the prototype was accelerated. Even in MATLAB, there were numerous challenges to getting a correct SMO implementation. At first, issues of numerical instability were a problem. It was necessary to explicitly enforce that certain

quantities could not become even slightly negative, since a change in sign could have disastrous effects. Once the basic problems were addressed, we discovered the need for additional "step types" as described in the preceding sections. Jebara [2004a] in its suggestion of SMO did not distinguish between these cases, and did not cover the entire feasible space. Even after we discovered the need for the intra-class step types, we had no knowledge of the need for the Newton steps. It took quite a bit of investigation, and in fact a few re-derivations, to convince ourselves that the algorithm was correct, because it did not always converge. Ultimately, it became apparent that the implementation was correct, but the application of a local optimization such as SMO was problematic. The realization came independently, but was supported by the claims in F. R. Bach [2004].

Once we had a correct MATLAB implementation, we translated it to C++ and used the MATLAB MEX interface to invoke the function. Again, we opted for correctness as opposed to speed in the first implementation. We wrote a C++ matrix class as a lightweight replacement for MATLAB's matrices. All necessary operations were implemented in C++, with support from LAPACK [Anderson et al., 1999] and BLAS [Lawson et al., 1979] (linear algebra and matrix libraries). Finally we went back and spent some time optimizing the implementation. To give some idea of the importance of code optimization, we show five plots (3.5.4) that were created as we progressed with the code modifications. Panel (f) of this figure shows our SMO compared against the highly optimized, MOSEK [Andersen and Andersen, 2000] commercial optimization software.

In optimizing the SMO software, some of the tasks that proved most valuable were moving costly, invariant code out of loops, replacing temporary object construction/destruction with static objects, and axis selection heuristics. Of these, the last is most interesting, because it is specific to SMO. Sequential *minimal* optimization involves choosing a minimal set of axes (variables) to update. The size of that set of axes is determined by the constraints, as described above. The question remains,

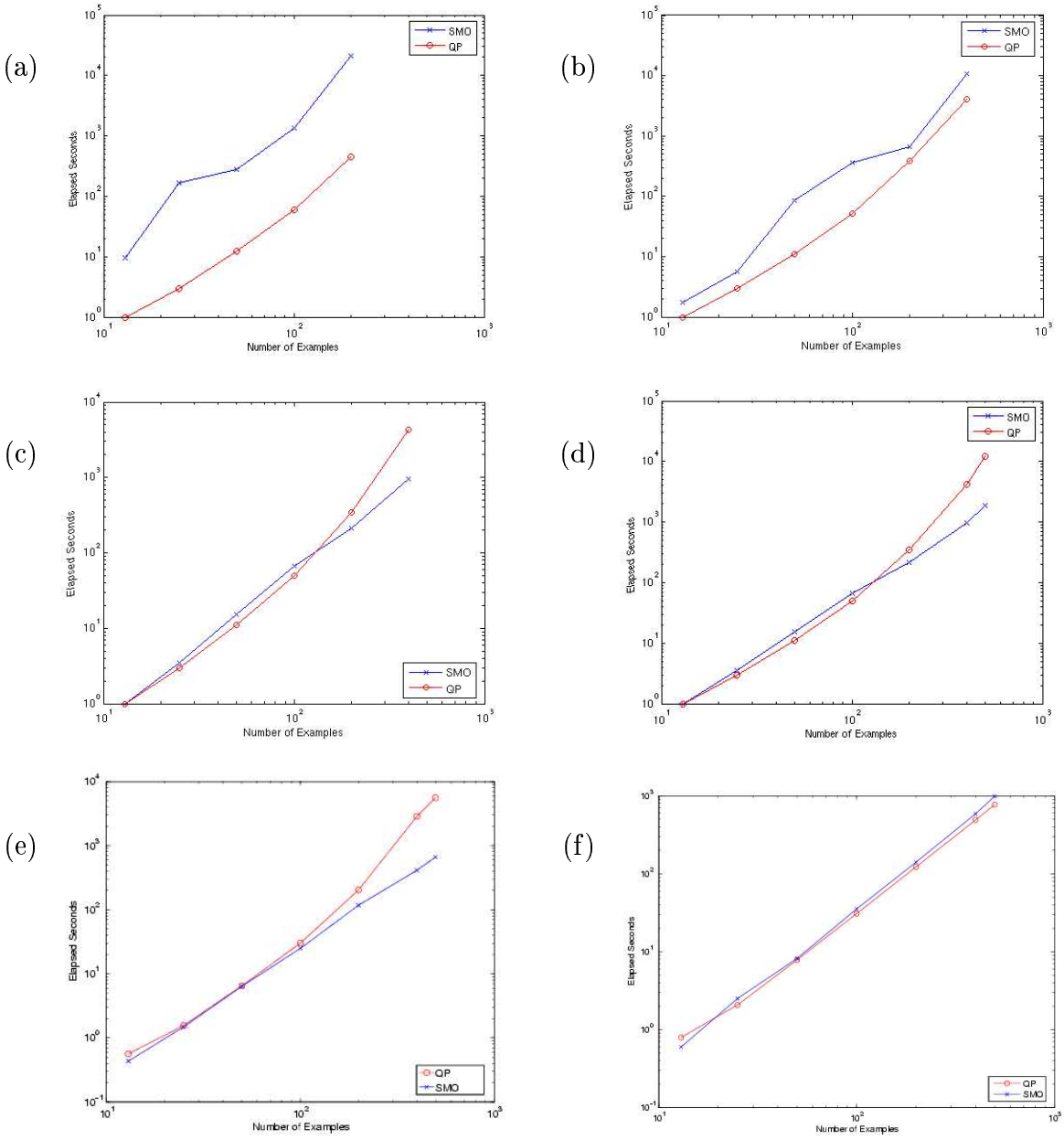


Figure 3.1: **SMO code optimization.** This figure shows the progression from the initial correct C++ implementation of the SMO algorithm, to the optimized version. (a-e) compare elapsed time for SMO and MATLAB `quadprog`. (f) compares elapsed time for SMO and MOSEK commercial optimization software.

which are the most suitable axes to choose. We call this problem *axis selection*. Platt [1999] uses heuristics that work well, on average, for the SVM optimization. Jebara [2004a, chap. 7] suggests "learning" axis transitions by greedily recalling past choices that made large improvements in the objective function. Here, we take something of a hybrid approach. We enumerate a random permutation of all non-redundant axis pairs. We iterate repeatedly over the permutation, removing pairs that produce little improvement, or that are at bound (0 or $c$). When the list becomes empty, we reinitialize it with a different random permutation. This algorithm provides good randomized coverage of the axis-pairs, but focuses effort on those that have performed well. The optimization is repeated until the change in solution and objective are bounded by a small threshold. Convergence is strictly monotonic. Our contribution is the first implementation of SMO for latent MED.

## 3.6 Nonstationary Kernel Combination

The kernelized latent MED mixture model can be viewed as a *nonstationary* kernel combination technique. Most methods for kernel combination assume that several base kernels are provided and a single convex combination is ultimately utilized. Using the above mixture modeling framework provides posteriors over the $\beta$ and $\alpha$ mixing proportions, which affect how much total influence each kernel has in the discriminant ratio. However, in addition, the input to the discriminant will determine a re-weighting for each Gaussian according to the Euclidean distance of each mapped input to each Gaussian mean in Hilbert space. This effectively re-weights the kernel combination in non-stationary ways and allows different kernel nonlinearities in the input space.

Therefore, this paper describes a technique for combining kernels that is more general than a weighted linear summation (a *stationary* combination). The technique uses a generative framework that specifically optimizes distributions to achieve max-

imal margins, effectively embedding latent variables into support vector machines. The approach thus permits the use of mixture modeling in a large-margin discriminative setting.

We demonstrate the power of the proposed technique by mixing kernels in a nonstationary fashion, allowing kernel weights to be a function of the data. In order to illustrate the differences among the standard SVM formulation, existing kernel combination methods, and the technique proposed here, consider the SVM discriminant function:

$$f(x) = \sum_{t \in \mathcal{T}} y_t \lambda_t k(x_t, x) + b.$$

Here, the class label assigned to example $x$ by the trained SVM depends on the training examples $x_t$, the SVM weights $\lambda_t$, a bias term $b$, and a single kernel function $k(\cdot, \cdot)$. In the SDP [Lanckriet et al., 2002] and hyperkernel [Ong et al., 2005] approaches, this formula is generalized to use multiple kernels, each of which has a fixed weight $\nu_m$:

$$f(x) = \sum_{t \in \mathcal{T}} y_t \lambda_t \sum_m \nu_m k_m(x_t, x) + b$$

The approach that we propose generalizes this formulation further, allowing the kernel weight $\nu_m$ to depend upon the properties of the example being classified:

$$f(x) = \sum_{t \in \mathcal{T}} y_t \lambda_t \sum_m \nu_{m,t}(x) k_m(x_t, x) + b$$

This dependence on $x$ is useful, for example, if the training data contains structure such that some types of examples are best classified on the basis of one kernel and other types of examples are best classified on the basis of a different kernel.

The decision rule in Equation (3.5) can be viewed as a nonstationary kernel combination. This is done by rewriting the equations using the definition of the expectations for the log-Gaussians. First, recall Jensen's inequality, which holds for any non-negative $q_m$ and $\nu_m$ scalars such that $\sum_m \nu_m = 1$:

$$\ln \sum_m q_m \;=\; \ln \sum_m \nu_m \frac{q_m}{\nu_m} \;\geq\; \sum_m \nu_m \ln \frac{q_m}{\nu_m}$$

If we set $\nu_m = \frac{q_m}{\sum_m q_m}$ and substitute into the right hand side above, the inequality is actually an equality:

$$\begin{aligned}
\ln \sum_m q_m &\geq \sum_m \frac{q_m}{\sum_m q_m} \ln \frac{q_m}{\frac{q_m}{\sum_m q_m}} \\
&\geq \sum_m \frac{q_m}{\sum_m q_m} \ln \sum_m q_m \\
&= \ln \sum_m q_m
\end{aligned}$$

The result is applied to the definition for the final decision rule (Equation (3.5)) to rewrite it as follows:

$$\begin{aligned}
\hat{y} = \sum_m \frac{\exp(E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+)\} + a_m)}{\sum_m \exp(E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+)\} + a_m)} E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+) + a_m\} \\
- \sum_n \frac{\exp(E\{\ln \mathcal{N}(\phi_n^-(X)|\mu_n^-)\} + b_n)}{\sum_n \exp(E\{\ln \mathcal{N}(\phi_n^-(X)|\mu_n^-)\} + b_n)} E\{\ln \beta_n \mathcal{N}(\phi_n^-(X)|\mu_n^-) + b_n\}
\end{aligned}$$

The ratios above can be written as the weights in a non-stationary kernel combination as follows:

$$\hat{y} = \sum_m \nu_m^+(X) E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+)\} + a_m - \sum_n \nu_n^-(X) E\{\ln \mathcal{N}(\phi_n^-(X)|\mu_n^-)\} + b_n$$

Recall that we have:

$$\begin{aligned}
&E\{\ln \mathcal{N}(\phi_m^+(X)|\mu_m^+)\} = \\
&\quad \sum_{\tau \in \mathcal{T}^+} \lambda_\tau q_\tau(m) k_m^+(X_\tau, X) - \sum_{\tau \in \mathcal{T}^-} \lambda_\tau Q_\tau(m) k_m^+(X_\tau, X) - \frac{1}{2} k_m^+(X, X) + constant
\end{aligned}$$

and that

$$\begin{aligned}
&E\{\ln \mathcal{N}(\phi_n^-(X)|\mu_n^-)\} = \\
&\quad \sum_{\tau \in \mathcal{T}^-} \lambda_\tau q_\tau(n) k_n^-(X_\tau, X) - \sum_{\tau \in \mathcal{T}^+} \lambda_\tau Q_\tau(n) k_n^-(X_\tau, X) - \frac{1}{2} k_n^-(X, X) + constant
\end{aligned}$$

Combining all terms yields:

$$\hat{y} = \sum_{\tau \in \mathcal{T}^+} \sum_m \lambda_\tau q_\tau(m) \nu_m^+(X) k_m^+(X_\tau, X) - \sum_{\tau \in \mathcal{T}^-} \sum_m \lambda_\tau Q_\tau(m) \nu_m^+(X) k_m^+(X_\tau, X)$$
$$- \sum_{\tau \in \mathcal{T}^-} \sum_n \lambda_\tau q_\tau(n) \nu_n^-(X) k_n^-(X_\tau, X) + \sum_{\tau \in \mathcal{T}^+} \sum_n \lambda_\tau Q_\tau(n) \nu_n^-(X) k_n^-(X_\tau, X)$$
$$+ \sum_m \nu_m^+(X) k_m^+(X, X) - \sum_n \nu_n^-(X) k_n^-(X, X) + constant,$$

from which we clearly see that the kernel combination weights depend on the input $X$, and the discriminant depends also on the learned parameters, which maximize the margin.

Nonstationary kernel combination is interesting, in that it displays more representational power than the linear combinations in popular use; however, our empirical studies indicate that it does not overfit. The latent MED prior distribution, in this case $\mathcal{N}(0, I)$ for the Gaussian means, regularizes the solution, as does the explicit soft margin bound constraint.

The following chapter presents a thorough set of experiments that compare NSKC to state-of-the-art techniques on a variety of problems.

# Chapter 4

# Nonstationary Kernel Combination Empirical Results

In this chapter, we present the experimental results we obtained with our nonstationary kernel combination technique. The chapter begins with two synthetic problems, designed primarily to illustrate the capabilities of the technique. The next section validates NSKC on some popular University of California at Irvine Machine Learning Repository [Murphy and Aha, 1995] benchmark data sets, which were chosen because they were used in Lanckriet et al. [2002]. Following those experiments, we discuss the application of NSKC to some real yeast protein function annotation problems.

## 4.1 Synthetic data sets

In this section, we validate the NSKC technique on two synthetic data sets. The primary purpose, here, is to illustrate the power of the technique to solve problems that existing approaches fail to solve. Thus, any positive result indicates a fundamental improvement over the state of the art. In all figures, the plotted decision boundaries were actually learned by the classifiers.

### 4.1.1 Eight-Gaussians data set

We use an MED ratio of two two-component Gaussian mixtures on eight Gaussian clusters to illustrate how margin optimization can overcome model mismatch. In this case, the margin optimization—absent from traditional maximum likelihood parameter estimation—drives the classifier toward the correct solution. Though latent MED optimizes a discriminative objective function, it does, learn a generative model. However, the model parameters are estimated to produce the most discriminative distribution. Therefore, generating data with the model will produce "ideal" samples with regard to discrimination. In this respect, we consider latent MED as a way to overcome model mismatch or to clean the data.

Figure 4.1(a) shows two independent two-component Gaussian mixtures fit to the data using maximum likelihood. Figure 4.1(b) shows latent MED with two-component Gaussian mixtures. This problem emphasizes a capability absent in the ML technique. In real data the effect would still be beneficial, but in smaller degree.

### 4.1.2 Linear-quadratic data set

In order to illustrate the characteristics of nonstationary kernel combination, we first present a synthetic two-dimensional problem. We generate a binary labeled data set using a function that is quadratic in part of the input space and linear elsewhere. The function is given by

$$f(x) = \begin{cases} x^2 & \text{if } |x| < 1, \\ 1 & \text{otherwise.} \end{cases}$$

Points are translated up and down along the vertical axis to create two classes for the binary problem. We then attempt to learn a decision surface using a maximum likelihood mixture of Gaussians (Figure 4.2a) and an SDP kernel combination (Figure 4.2b). While the SDP result may seem surprising at first, consider the objective the SDP is optimizing. The linear kernel combination weights are chosen along with

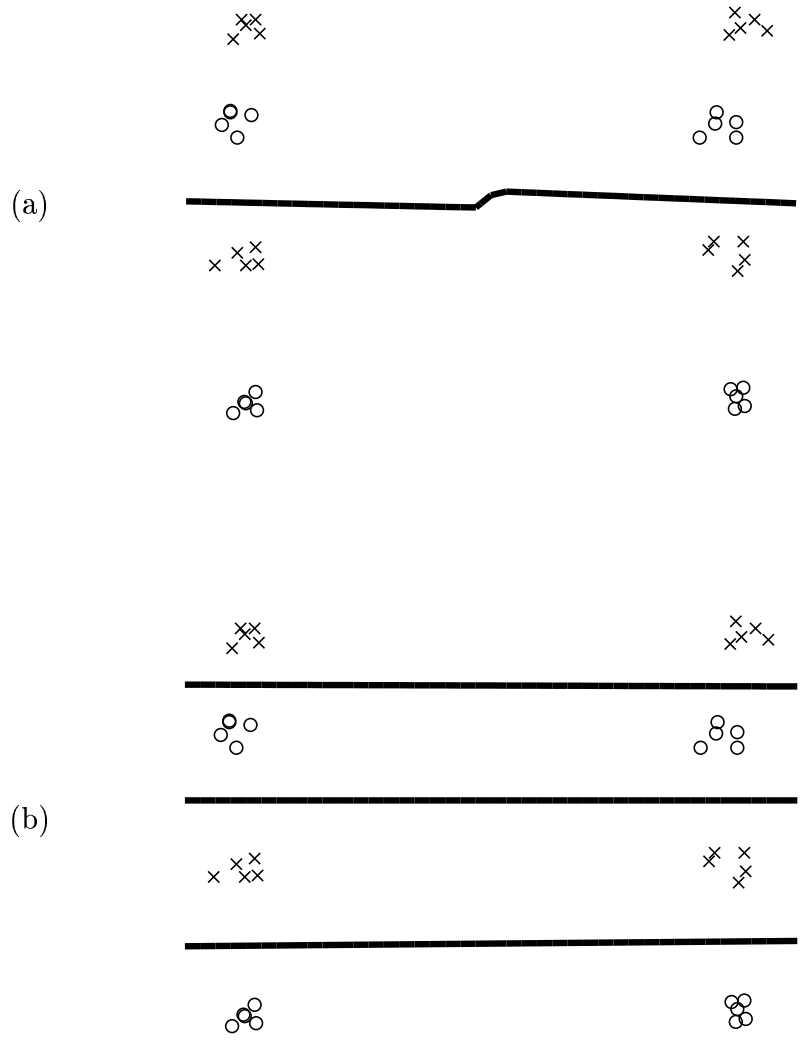


Figure 4.1: **Latent MED Gaussian mixture.** This figure shows the ability of latent MED to learn a large margin classifier, even in an extreme case of model mismatch. The classifier is the ratio of two Gaussian mixtures, each with two components. Maximum likelihood (a) classifies poorly, achieving 50% accuracy because it ignores discriminative performance. Latent MED (b) achieves perfect classification accuracy.

the Lagrange multipliers to minimize loss and maximize margin. That is precisely what is accomplished.

Figure 4.3(a) shows the classifier with the proposed nonstationary kernel combination. Note that NSKC achieves perfect separation of the data with maximum margin. The decision surface is extremely smooth except at the transition between linear and quadratic regions. One may argue that in this example, an RBF kernel could learn a similar decision surface. However, the RBF solution would behave poorly in regions of low density. The RBF is a higher capacity function and would not likely produce as smooth a solution. Also, in real applications, the kernel often encodes some domain knowledge, which is preserved by NSKC.

Figure 4.3(b) shows how the combination weights vary over the input space. This illustrates the nonstationarity that we have described and derived. This example exhibits the novelty of the NSKC technique and indicates its usefulness. In any situation in which the combination weights man need to vary over the input space, NSKC will have an advantage.

## 4.2 Benchmark data sets

Next, we validate the nonstationary kernel combination technique on several benchmark data sets from the UCI machine learning repository. We use the Wisconsin breast cancer, sonar, and statlog heart data, all of which were previously used to validate the SDP method [Lanckriet et al., 2002]. We use three kernels: a quadratic kernel

$$k_1(x_1, x_2) = (1 + x_1^T x_2)^2,$$

a radial basis function (RBF) kernel

$$k_2(x_1, x_2) = \exp(-0.5(x_1 - x_2)^T(x_1 - x_2)/\sigma),$$

and a linear kernel

$$k_3(x_1, x_2) = x_1^T x_2.$$

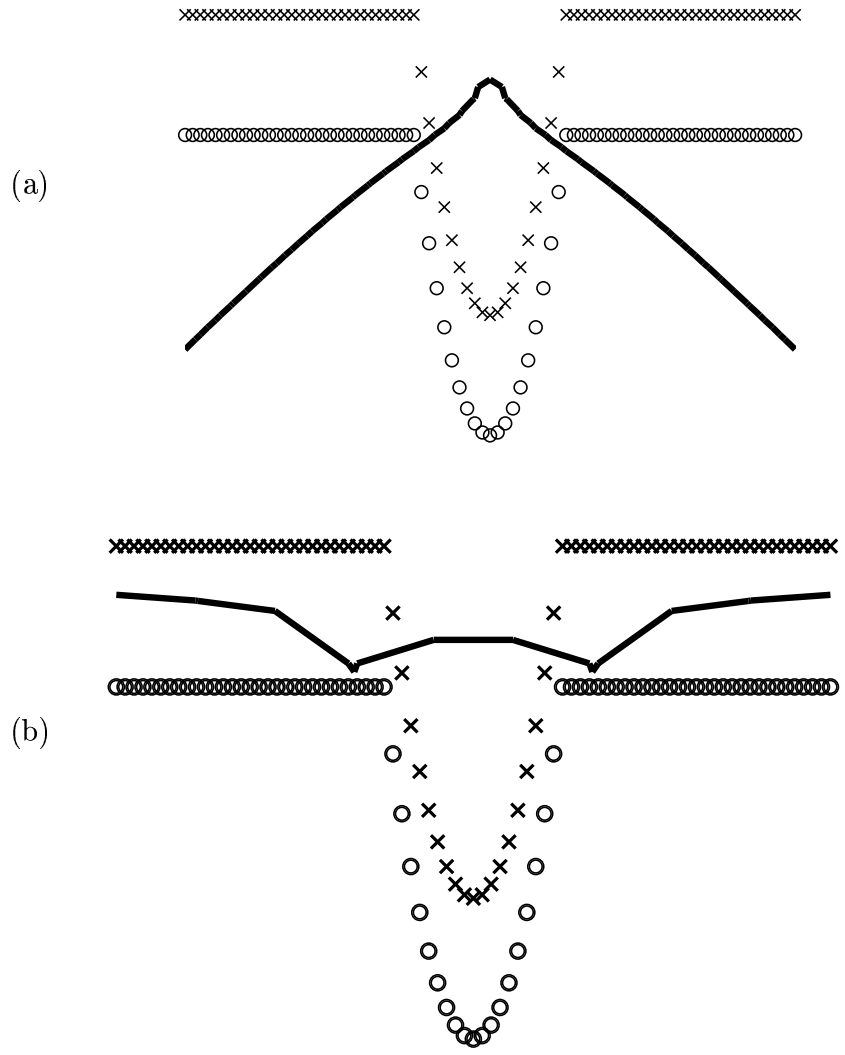


Figure 4.2: **Kernel combination on synthetic data.** The figure illustrates the binary decision surface between examples taken from a function that is linear and quadratic. Panel (a) shows the ML mixture decision boundary; panel (b) shows the SDP decision boundary. Neither technique correctly classifies the data.

(a)

(b)

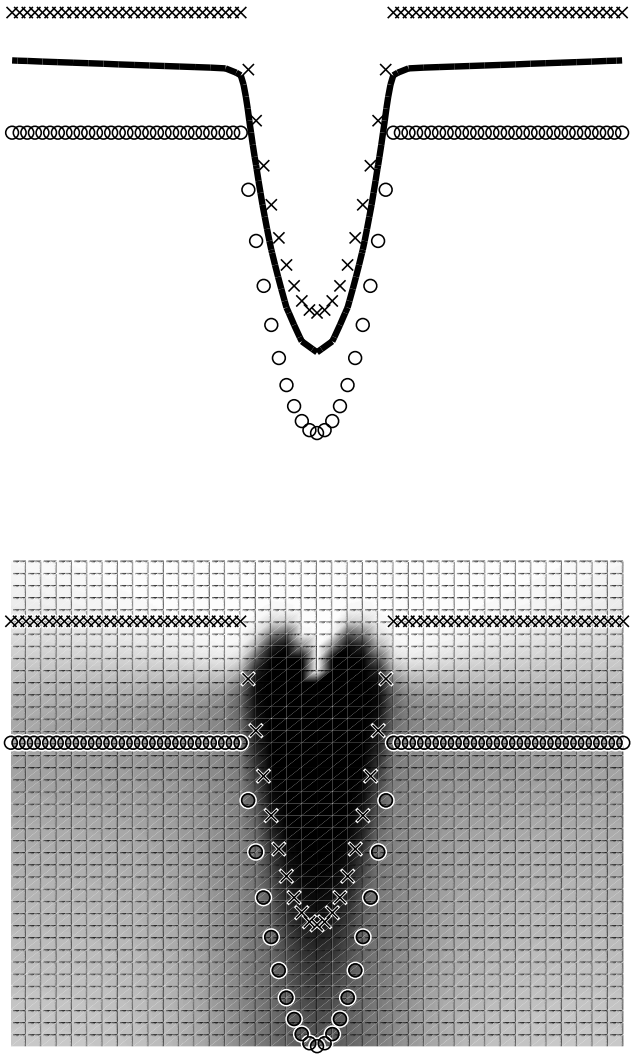

Figure 4.3: **Nonstationary kernel combination on synthetic data.** The figure illustrates the binary decision surface between examples taken from a function that is piecewise linear and quadratic. Panel (a) shows the NSKC decision boundary; NSKC correctly separates the data. Panel (b) shows the NSKC kernel weight over the input space; darker shades correspond to the quadratic kernel. Note the smoothness of the solution.

All three kernels are normalized so that their features lie on the surface of a unit hypersphere, $\hat{k}(x,z) = \frac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}$. As in [Lanckriet et al., 2002], we use a hard margin (with $c = 10,000$ to avoid numerical difficulties when $c = \infty$), and the RBF width parameter $\sigma$ is set to 0.5, 0.1 and 0.5, respectively, for the three tasks. NSKC and maximum likelihood (ML) mixtures use one mixture component per kernel for each class model, i.e., $M = N = 3$. We report mean test set accuracy across five random replications of three-fold cross validation.

Tables 4.1, 4.2, and 4.3 summarize the results of these experiments. NSKC achieves the top mean test set accuracy for two of three data sets. For the heart data set, NSKC is second best. We believe that this good generalization performance is due, in large part, to NSKC's ability to achieve a more regularized classifier by using the lower capacity polynomial and linear kernels for regions of the input space in which they perform well. By contrast, the SDP technique is forced to down-weight those entire kernels and use the RBF kernel with hard margin. With SDP, we observed an overfit classifier in many cases, i.e., train accuracy was often perfect while test accuracy was not.

### 4.2.1 Wisconsin breast cancer

The breast cancer data set is not separable using the polynomial or linear kernels. Nonetheless, these kernels are useful to the combination techniques. Indeed, both the SDP and NSKC methods gain significant advantage by combining kernels (Table 4.1).

### 4.2.2 Sonar

The sonar data set is the only data set that is separable using each of the three kernels individually. The SDP technique matches the accuracy of the best individual kernel, the RBF. NSKC achieves a small additional gain from nonstationarity (Table 4.2).

Table 4.1: **Comparison on Wisconsin breast cancer data.** The table lists, for the UCI Wisconsin breast cancer data set and six classification methods, the mean and standard deviation of test set accuracy across fifteen cross-validations (three-fold CV repeated five times). The first three methods are SVMs trained with single kernels, followed by the SDP approach of [Lanckriet et al., 2002], a maximum likelihood mixture of Gaussians classifier, and the NSKC method. The maximal mean value is indicated in boldface.

| Algorithm | Mean ROC |
|---|---|
| quadratic | $0.5486 \pm 0.091$ |
| RBF | $0.6275 \pm 0.019$ |
| linear | $0.5433 \pm 0.087$ |
| SDP | $0.8155 \pm 0.015$ |
| ML | $0.5573 \pm 0.03$ |
| NSKC | $\mathbf{0.8313 \pm 0.014}$ |

Table 4.2: **Comparison on sonar data.** The table lists, for the UCI sonar data set and six classification methods, the mean and standard deviation of test set accuracy across fifteen cross-validations (three-fold CV repeated five times). The first three methods are SVMs trained with single kernels, followed by the SDP approach of [Lanckriet et al., 2002], a maximum likelihood mixture of Gaussians classifier, and the NSKC method. The maximal mean value is indicated in boldface.

| Algorithm | Mean ROC |
|---|---|
| quadratic | $0.8145 \pm 0.01$ |
| RBF | $0.8595 \pm 0.009$ |
| linear | $0.7297 \pm 0.01$ |
| SDP | $0.8595 \pm 0.009$ |
| ML | $0.6817 \pm 0.022$ |
| NSKC | $\mathbf{0.8634 \pm 0.008}$ |

### 4.2.3 Heart

For the heart data set, the maximum test set accuracy is achieved when using the polynomial of degree two, with which the data set is not separable. The data is separable using the kernel combination techniques; however, neither SDP nor NSKC is able to make full use of the polynomial kernel. NSKC is apparently able to use the polynomial over at least part of the input space (Table 4.3).

## 4.3 Yeast protein functional classification

In a larger experiment, we apply our nonstationary kernel combination to the problem of protein function annotation from a collection of heterogeneous kernels. We compare the latent MED method against SVMs using single kernels and the SDP method in [Lanckriet et al., 2004b] using data from that study (`noble.gs.washington.edu/`

Table 4.3: **Comparison on heart data.** The table lists, for the UCI heart data set and six classification methods, the mean and standard deviation of test set accuracy across fifteen cross-validations (three-fold CV repeated five times). The first three methods are SVMs trained with single kernels, followed by the SDP approach of [Lanckriet et al., 2002], a maximum likelihood mixture of Gaussians classifier, and the NSKC method. The maximal mean value is indicated in boldface.

| Algorithm | Mean ROC |
|---|---|
| quadratic | **0.6141 ± 0.032** |
| RBF | 0.5556 ± 0.01 |
| linear | 0.5237 ± 0.02 |
| SDP | 0.5556 ± 0.01 |
| ML | 0.5361 ± 0.024 |
| NSKC | 0.6052 ± 0.016 |

`proj/yeast`). We use three kernels, representing gene expression, protein domain content, and protein sequence similarity. We train one-versus-all classifiers for 12 functional classes of yeast genes. We randomly sample from the data set to reduce its size to 500 genes and then perform three-fold cross-validation, repeating the entire procedure five times. For all methods, we use the regularization parameter $c = 10$, as in [Lanckriet et al., 2004b]. With NSKC, we again use one component per kernel per class in the model.

Table 4.4 summarizes the mean AUCs over 15 trials for all methods. The NSKC method has the highest accuracy for eight of the twelve classes. We speculate that an advantage for NSKC over SDP in these experiments is the ability of the mixture model to capture some of the latent structure in the data. In particular, the 12 yeast functional classes represent the top level of the MIPS functional hierarchy. Thus, these 12 classes contain meaningful substructure, which NSKC can exploit.

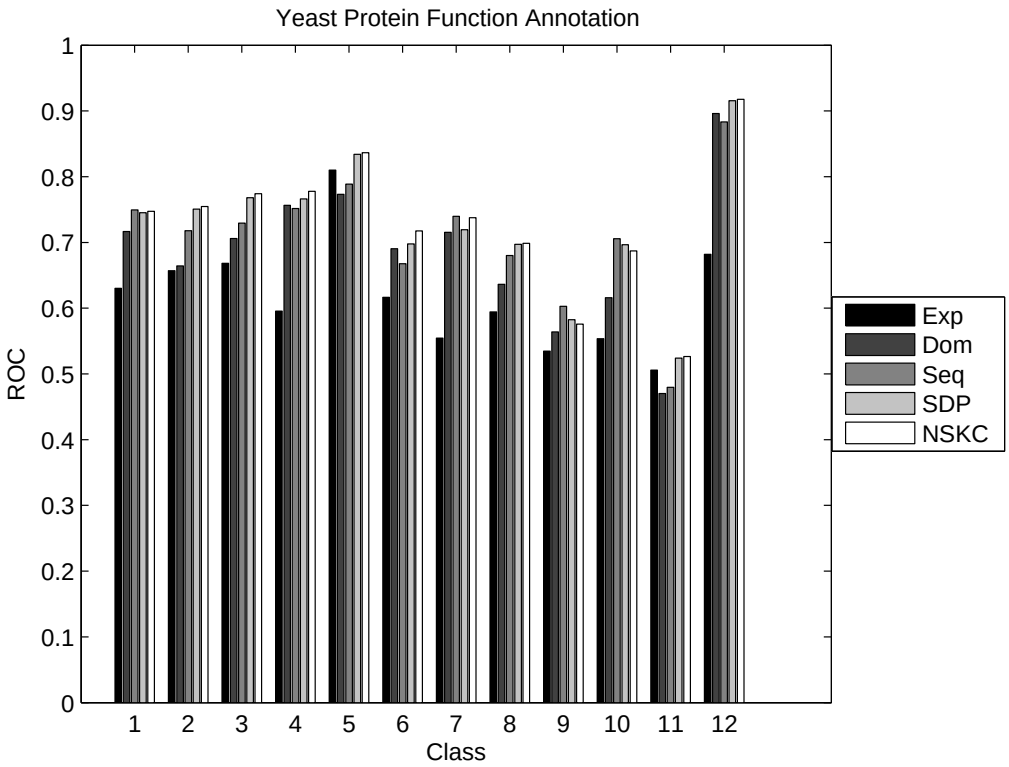


Figure 4.4: **Nonstationary kernel combination for yeast protein function annotation.** The figure shows, for each functional class and each classification method the mean AUC from five times three-fold cross-validation.

Table 4.4: **Comparison of yeast protein function annotation methods.** The table lists, for each functional class (row) and each classification method (column) the mean AUC from five times three-fold cross-validation. The first three columns correspond to SVMs trained on single kernels (gene expression, protein domain content and sequence similarity, respectively). The final two columns contain results for the SDP and nonstationary kernel combination methods. For all methods, standard errors (not shown) are generally on the order of 0.02, except for classes 2 (0.04) and 9 (0.05).

| Class | Exp | Dom | Seq | SDP | NSKC |
|---|---|---|---|---|---|
| 1 | 0.630 | 0.717 | **0.750** | 0.745 | 0.747 |
| 2 | 0.657 | 0.664 | 0.718 | 0.751 | **0.755** |
| 3 | 0.668 | 0.706 | 0.729 | 0.768 | **0.774** |
| 4 | 0.596 | 0.756 | 0.752 | 0.766 | **0.778** |
| 5 | 0.810 | 0.773 | 0.789 | 0.834 | **0.836** |
| 6 | 0.617 | 0.690 | 0.668 | 0.698 | **0.717** |
| 7 | 0.554 | 0.715 | **0.740** | 0.720 | 0.738 |
| 8 | 0.594 | 0.636 | 0.680 | 0.697 | **0.699** |
| 9 | 0.535 | 0.564 | **0.603** | 0.582 | 0.576 |
| 10 | 0.554 | 0.616 | **0.706** | 0.697 | 0.687 |
| 11 | 0.506 | 0.470 | 0.480 | 0.524 | **0.526** |
| 12 | 0.682 | 0.896 | 0.883 | 0.916 | **0.918** |

Table 4.5: **Standard error for yeast mean AUC scores.** The table lists, for each functional class (row) and each classification method (column) the standard errors associated with the corresponding means from Table 4.4.

| Class | Exp | Dom | Seq | SDP | NSKC |
|---|---|---|---|---|---|
| 1 | 0.02 | 0.01 | 0.009 | 0.014 | 0.016 |
| 2 | 0.037 | 0.029 | 0.038 | 0.041 | 0.042 |
| 3 | 0.03 | 0.018 | 0.016 | 0.014 | 0.013 |
| 4 | 0.015 | 0.012 | 0.02 | 0.019 | 0.015 |
| 5 | 0.024 | 0.017 | 0.019 | 0.019 | 0.02 |
| 6 | 0.018 | 0.017 | 0.011 | 0.014 | 0.015 |
| 7 | 0.017 | 0.016 | 0.015 | 0.019 | 0.016 |
| 8 | 0.02 | 0.031 | 0.021 | 0.016 | 0.018 |
| 9 | 0.02 | 0.056 | 0.057 | 0.057 | 0.055 |
| 10 | 0.021 | 0.024 | 0.019 | 0.023 | 0.026 |
| 11 | 0.041 | 0.031 | 0.032 | 0.022 | 0.03 |
| 12 | 0.03 | 0.013 | 0.009 | 0.007 | 0.008 |

Table 4.6: **Comparison of yeast protein function annotation methods using sequence and structure.** The table lists, for each GO term (row) and each classification method (column) the mean AUC from three times five-fold cross-validation. The first column corresponds to a kernel average, the second to SDP, and the third to nonstationary kernel combination.

| GO term | Average | SDP | NSKC |
|---|---|---|---|
| GO:0008168 | $0.937 \pm 0.016$ | $0.938 \pm 0.015$ | $0.944 \pm 0.014$ |
| GO:0005506 | $0.927 \pm 0.012$ | $0.927 \pm 0.012$ | $0.926 \pm 0.013$ |
| GO:0006260 | $0.878 \pm 0.016$ | $0.870 \pm 0.015$ | $0.880 \pm 0.015$ |
| GO:0048037 | $0.911 \pm 0.016$ | $0.909 \pm 0.016$ | $0.918 \pm 0.015$ |
| GO:0046483 | $0.937 \pm 0.008$ | $0.940 \pm 0.008$ | $0.941 \pm 0.008$ |
| GO:0044255 | $0.874 \pm 0.015$ | $0.864 \pm 0.013$ | $0.874 \pm 0.012$ |
| GO:0016853 | $0.837 \pm 0.017$ | $0.810 \pm 0.019$ | $0.823 \pm 0.018$ |
| GO:0044262 | $0.908 \pm 0.006$ | $0.897 \pm 0.006$ | $0.906 \pm 0.007$ |
| GO:0009117 | $0.890 \pm 0.012$ | $0.880 \pm 0.012$ | $0.887 \pm 0.012$ |
| GO:0016829 | $0.931 \pm 0.008$ | $0.926 \pm 0.007$ | $0.928 \pm 0.008$ |

Table 4.6 summarizes another experiment we conducted for yeast protein function annotation. This time, we use the sequence/structure dataset from Chapter 2. The mean AUCs over 15 trials for kernel averaging, SDP, and NSKC are listed. There is no clear winner between NSKC and kernel averaging; however, NSKC consistently outperforms the SDP technique. We computed $p$-values using a one-sided Wilcoxon signed rank test that show a significant advantage for NSKC over SDP. Again, we find that averaging is surprisingly good with this data. Statistically, averaging equals NSKC over these results.

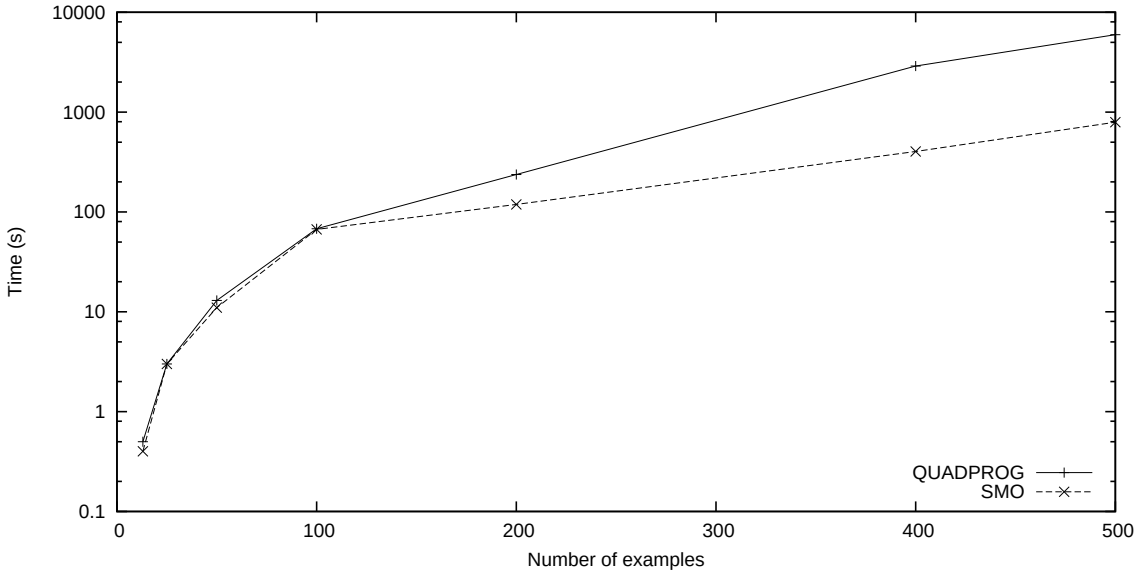


Figure 4.5: **Running time comparison of SMO and `quadprog`.** The figure plots running time as a function of the number of examples for our SMO implementation and MATLAB's `quadprog` on the toy data set described in Section 4.1.2.

## 4.4 SMO Timing

In order to evaluate the time complexity of our optimization procedure, we ran a small timing experiment using the synthetic data set described in Section 4.1.2. We computed the latent MED discriminative projection step for varying numbers of examples. We repeated this procedure several times to compute mean optimization times for our SMO and for MATLAB's `quadprog` function. The results are summarized in Figure 4.5. With 500 examples, SMO runs 7.5 times faster than `quadprog`.

# Chapter 5

# Conclusion

## 5.1 Summary of contribution

We set out to extend the domain of large margin classification [Vapnik, 1998] and kernel methods [Schölkopf et al., 1999] to the realm of latent graphical models, and in so doing, discovered a novel way to combine kernels in a multi-kernel learning algorithm. We call this approach nonstationary kernel combination (NSKC), because the kernel combination weights vary over the input space, rather than being a constant, stationary distribution. NSKC generalizes upon existing kernel combination techniques [Lanckriet et al., 2002, Ong et al., 2005], defining the combination weights using the posterior distribution of a mixture model, rather than a simple linear combination. The mixture model adds capacity, in the VC sense [Vapnik, 1998], to the resulting classifier. While increased capacity gives the classifier more power to overfit, the regularization imparted by minimum relative entropy parameter estimation along with a sensible prior distribution encourages a smooth solution that generalizes well. It is important to note that the increased capacity takes the form of a structured generative model. This is not as arbitrary as increasing capacity with a more general kernel function, e.g., an RBF kernel. NSKC preserves the prior knowledge given in the kernels, but allows more flexibility in its use.

We proposed a more principled latent MED variational bounding technique than Jebara [2004a] that achieves a tightly bounded locally optimal solution. This was achieved by a second application of the Jensen's inequality bound to eliminate the intractability resulting from the incorrect-class log-sums that remain after the first application of Jensen's inequality. The original intractable partition function is sandwiched from below and above between the bounding terms of the new tractable partition function. When both bounds are tightened via minimax optimization, the original partition is recovered. We combine the bound tightening steps with a maximization over the objective, to iteratively converge to a tightly bounded locally optimal solution. We presented detailed derivations for the latent MED mixture of Gaussians and explained the general case in sufficient detail for others to extend the work.

We also fully explained our approach to efficient optimization through SMO [Platt, 1999]. We began with the linear system of constraints and showed how to ensure that the constraints remain satisfied, i.e., that we remain in the null space of the linear system, while modifying as few axes as possible. We discovered three different cases for SMO step types and explicated the derivations to obtain the correct update equations. We also shared our experience in applying SMO to latent MED problems, and the difficulty that we faced in obtaining consistent convergence. The latent MED objective is convex, but not smooth. Thus, it is possible to become "trapped" on a locally flat region of the objective and make virtually no progress toward the minimum with each step. In these cases, we found it necessary to take a bigger step toward the global minimum by considering all active axes simultaneously with a constrained Newton step. The Newton step does not destroy the computational efficiency of the technique, because the number of active axes involved in the inversion of the KKT system—a $O(n^3)$ operation—tends to be less than $n^{\frac{2}{3}}$. Therefore, the time required for the SVD to compute the null space does not exceed $O(n^2)$ over all axes, the average complexity of the SMO algorithm.

We conducted several empirical studies using kernel combination, including one set of experiments [Lewis et al., 2006c] that investigates the cost/benefit trade-off of learning kernel combination weights via optimization versus using an unweighted combination. This study is particularly relevant to practitioners due to the time required to compute the combination weights for large data sets and due to the current lack of freely available software to perform the optimization. We determined that unweighted combination can be as accurate as an optimized combination. This study also examined the effect of missing data with kernel combination. We found that the simple unweighted combination is quite robust to missing data. In another experimental direction, we validated the NSKC technique on numerous problems and have found that in the majority of cases, results have been better than existing techniques. These results did not require particular tuning of the algorithm.

We have created a complete MATLAB/C++ software implementation, including MATLAB classes for kernel functions, learning algorithms, and, cross validation experiments. We provide an implementation of the SMO optimization as a MATLAB MEX file written in C++. We will make the software publicly available for academic use. A user guide to to the software is provided in Appendix A.

## 5.2 Future Directions

In formulating the latent MED iterative bounding and optimization technique, we chose a very EM-like [Dempster et al., 1977] algorithm. The M-step, which updates the model parameters, is discriminative and involves a convex optimization. However, the E-step only involves computing simple expectations given the model and essentially ignores the discriminative goal. The technique works, but may suffer from slower convergence or local minima than if we actively optimized both projection steps. Latent MED can be formulated as a saddle-point optimization problem of an indefinite objective function. The solution would be found with a minimax optimiza-

tion. We did derive latent MED in this way; however, we did not find an efficient way to update the required distributions. There was some hope of finding a tractable solution, and this could be a direction for future research.

In addition, we simplified the latent MED objective by omitting the non-quadratic entropy terms associated with the $Q$ variational distributions. This was justified by the fact that the converged distributions tend to be like delta functions with all weight on a single component. Therefore, the entropy tends toward zero, and the objective truly becomes quadratic. However, at early stages of the latent MED optimization, the distributions are not fully committed, and the addition of the entropy terms would likely help the optimization to proceed more smoothly and avoid commitment to a local minima. This might ultimately speed the convergence to a more globally optimal solution and prevent cyclic behavior, which is sometimes seen with the current implementation. It should be relatively easy to add the entropy terms to the SMO optimization or to add the entropy terms to the objective and optimize the second-order Taylor approximation of the objective using QP.

Our experiments showed NSKC results that usually outperformed SDP kernel combination. However, the SDP algorithm [Lanckriet et al., 2002] had a clear advantage for these experiments because it is a transductive technique, i.e., SDP used the test examples when learning the combined kernel. The SDP formulation computes the regularizing trace constraint on the entire combined the kernel matrix, so smoothness is encouraged over the test data. Our current NSKC algorithm did not use the test examples, though it could have. The latent MED distribution can easily be augmented with a latent variable for the label of the test data, and we can integrate over that variable. Since probabilistic models handle semi-supervised classification naturally, and this area has been studied, this would be a beneficial extension to NSKC.

Though we covered the application of kernel combination quite thoroughly, the formalism of latent MED is more general and can support other probabilistic mod-

els. The next logical model to explore would be the hidden Markov model (HMM) [Rabiner, 1995]. Essentially, the HMM with Gaussian emissions is just the mixture model we already have with non-iid distribution over the multinomial latent variable. The Markov assumption for a first-order model is that the future is independent of the past, given the present. Thus, the mixing proportions for the next sample depend on the current latent state. Latent MED with HMMs is the most ambitious of the tasks listed here. The HMM involves more multinomial distributions, which introduces more Dirichlet priors, and more optimization constraints. Although this work would constitute a new project, we believe it would be worthwhile future work and would build on the current research.

# Bibliography

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden markov support vector machines. In *20th International Conference on Machine Learning (ICML)*, 2003.

E. D. Andersen and K. D. Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. *High Performance Optimization*, pages 197–232, 2000.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

F. R. Bach, R. Thibaux, and M. I. Jordan. Computing regularization paths for learning multiple kernels. In Lawrence K. Saul, Yair Weiss, and Leon Bottou, editors, *Advances in Neural Information Processing Systems*, Cambridge, MA, 2004. MIT Press.

A. Ben-Hur and W. S. Noble. Kernel methods for predicting protein-protein interactions. *Bioinformatics*, 21 suppl 1:i38–i46, 2005.

K.M. Borgwardt, C. S. Ong, S. Schoenauer, S.V.N. Vishwanathan, A. Smola, and H-P. Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 (Suppl. 1):i47–i56, 2005.

B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Prentice-Hall, 2003. To appear. Available at http://www.stanford.edu/~boyd/cvxbook.html.

R. Collobert, F. Sinz, J. Weston, and L. Bottou. Large scale transductive SVMs. In preparation, November 2005.

C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.

N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge UP, Cambridge, UK, 2000.

N. Cristianini, J. Shawe-Taylor, A. Elisseef, and J. Kandola. On kernel-target alignment. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–22, 1977.

H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines. In *Advances in Neural Information Processing Systems*, pages 155–161, Cambridge, MA, 1997. MIT Press.

S. Dumais. Using svms for text categorization. *IEEE Intelligent Systems Magazine*, 13(4), 1998.

M. I. Jordan F. R. Bach, G. R. G. Lanckriet. Multiple kernel learning, conic duality, and the smo algorithm. In *21st International Conference on Machine Learning (ICML)*, 2004.

A. Gammerman, V. Vovk, and V. Vapnik. Learning by transduction. In G. F. Cooper and S. Moral, editors, *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 148–155, San Francisco, CA, 1998. Morgan Kaufmann.

Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nature Genetics*, 25(1):25–29, 2000.

Federico Girosi, Michael Jones, and Tomaso Poggio. Regularization theory and neural networks architectures. *Neural Computation*, 7(2):219–269, 1995.

L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123–138, 1993.

T. Jaakkola, M. Diekhans, and D. Haussler. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 149–158, Menlo Park, CA, 1999a. AAAI Press.

T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems 11*, San Mateo, CA, 1998. Morgan Kauffmann.

T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. Technical Report AITR-1668, Massachusetts, December 1999b.

T. Jaakkola, M. Meila, and T. Jebara. Maximum entropy discrimination. In *Advances in Neural Information Processing Systems*, volume 12, December 1999c.

T. Jebara. *Machine Learning: Discriminative and Generative*. Kluwer Academic, Boston, MA, 2004a.

T. Jebara. Multi-task feature and kernel selection for SVMs. In *Proceedings of the International Conference on Machine Learning*, 2004b.

T. Jebara and A. Pentland. Maximum conditional likelihood via bound maximization and the cem algorithm. In *Advances in Neural Information Processing Systems 11*, San Mateo, CA, 1998. Morgan Kauffmann.

T. Joachims. Making large-scale support vector machine learning practical. In A. Smola B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

T. Joachims. Transductive inference for text classification using support vector machines. In I. Bratko and S. Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 200–209, San Francisco, CA, 1999. Morgan Kaufmann.

T. Joachims, N. Cristianini, and J. Shawe-Taylor. Composite kernels for hypertext categorisation. In C. Brodley and A. Danyluk, editors, *Proceedings of the International Conference on Machine Learning*, pages 250–257, San Francisco, 2001. Morgan Kaufmann.

M. Jordan and Bishop. An introduction to graphical models. In press.

M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. Introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy. Improvements to platt's smo algorithm for svm classifier design, 1999.

G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004a.

G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semi-definite programming. In C. Sammut and A. Hoffman, editors, *Proceedings of the 19th International Conference on Machine Learning*, Sydney, Australia, 2002. Morgan Kauffman.

G. R. G. Lanckriet, M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble. Kernel-based data fusion and its application to protein function prediction in yeast. In R. B. Altman, A. K. Dunker, L. Hunter, T. A. Jung, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 300–311. World Scientific, 2004b.

C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran usage. *ACM Transactions on Mathematical Software*, 5 (3):308–323, September 1979. ISSN 0098-3500.

C. Leslie, E. Eskin, and W. S. Noble. The spectrum kernel: A string kernel for SVM protein classification. In R. B. Altman, A. K. Dunker, L. Hunter, K. Lauderdale, and T. E. Klein, editors, *Proceedings of the Pacific Symposium on Biocomputing*, pages 564–575, New Jersey, 2002. World Scientific.

C. Leslie, E. Eskin, J. Weston, and W. S. Noble. Mismatch string kernels for SVM protein classification. In Suzanna Becker, Sebastian Thrun, and Klaus Obermayer, editors, *Advances in Neural Information Processing Systems*, pages 1441–1448, Cambridge, MA, 2003. MIT Press.

D. Lewis, T. Jebara, and W. S. Noble. Large margin classification using exponential family mixtures. Submitted, April 2006a.

D. Lewis, T. Jebara, and W. S. Noble. Nonstationary kernel combination. In *23rd International Conference on Machine Learning (ICML)*, 2006b.

D. Lewis, T. Jebara, and W. S. Noble. Support vector machine learning from heterogeneous data: an empirical analysis using protein sequence and structure. Submitted, April 2006c.

W. Li, L. Jeroszewski, and A. Godzik. Clustering of highly homologous sequences to reduce the size of large protein databases. *Bioinformatics*, 17(3):282–283, 2001.

L. Liao and W. S. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *Proceedings of the Sixth Annual International Conference on Computational Molecular Biology*, pages 225–232, Washington, DC, April 18–21 2002.

M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1967.

M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.

P. M. Murphy and D. W. Aha. UCI repository of machine learning databases. Dept. of Information and Computer Science, UC Irvine, 1995.

W. S. Noble. Support vector machine applications in computational biology. In J.-P. Vert B. Schoelkopf, K. Tsuda, editor, *Kernel methods in computational biology*, pages 71–92. MIT Press, Cambridge, MA, 2004.

C. S. Ong, A. J. Smola, and R. C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6:1043–1071, 2005.

A. R. Ortiz, C. E. M. Strauss, and O. Olmea. MAMMOTH (Matching molecular models obtained from theory): An automated method for model comparison. *Protein Science*, 11:2606–2621, 2002.

E. Osuna, R. Freund, and F. Girosi. Training support vector machines:an application to face detection, 1997.

P. Pavlidis, I. Wapinski, and W. S. Noble. Support vector machine classification on the web. *Bioinformatics*, 20(4):586–587, 2004.

P. Pavlidis, J. Weston, J. Cai, and W. N. Grundy. Gene functional classification from heterogeneous data. In *Proceedings of the Fifth Annual International Conference on Computational Molecular Biology*, pages 242–248, 2001.

P. Pavlidis, J. Weston, J. Cai, and W. S. Noble. Learning gene functional classifications from multiple data types. *Journal of Computational Biology*, 9(2):401–411, 2002.

J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, 1998.

J. C. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods*. MIT Press, 1999.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1995.

F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1959.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. pages 318–362, 1986.

B. Schölkopf, P. L. Bartlett, A. Smola, and R. Williamson. Shrinking the tube: a new support vector regression algorithm. In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, pages 330–336, Cambridge, MA, 1999. MIT Press.

B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1999.

B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

I. N. Shindyalov and P. E. Bourne. Protein structure alignment by incremental combinatorial extension (ce) of the optimal path. *Protein Engineering*, 11:739–747, 1998.

T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

S. Sonnenburg, G Rätsch, and C. Schafer. A general and efficient multiple kernel learning algorithm. In *Advances in Neural Information Processing Systems*, 2006a.

S. Sonnenburg, G Rätsch, and C. Schafer. Learning interpretable SVMs for biological sequence classification. *BMC Bioinformatics*, 7(Suppl. 1):S1–S9, 2006b.

B. Taskar, C. Guestrin, and D. Koller. Max margin markov networks. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, Cambridge, MA, 2003. MIT Press.

I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. *Proceedings of the International Conference on Machine Learning*, 2004.

K. Tsuda. Support vector classification with asymmetric kernel function. In M. Verleysen, editor, *Proceedings ESANN*, pages 183–188, 1999.

K. Tsuda, H.J. Shin, and B. Schölkopf. Fast protein classification with multiple networks. In *ECCB*, 2005.

V. N. Vapnik. *Statistical Learning Theory.* Adaptive and learning systems for signal processing, communications, and control. Wiley, New York, 1998.

P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science.* PhD thesis, Harvard University, Cambridge, MA, 1974.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. In Sara A Solla, Todd K Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 13*. MIT Press, 2001. URL `http://www.ens-lyon.fr/~ochapell/feature_sel.ps`.

J. Weston and C. Watkins. Multi-class support vector machines. *Royal Holloway Technical Report CSD-TR-98-04*, 1998.

A. Zien and C. S. Ong. An automated combination of sequence motif kernels for predicting protein subcellular localization. Technical Report TR-146, Max-Planck Institute for Biological Cybernetics, Tübingen, Germany, April 2006.

# Appendices

# Appendix A

# MaLT User Guide

Part of the contribution of this thesis is the software developed to validate the proposed techniques. We disseminate the software for academic use to encourage the adoption and extension of latent MED.

The software will inevitably continue to evolve after the deposit of this thesis. However, the thesis will remain an archival document. Therefore, a current version of this appendix will be distributed with the software. This appendix serves to clarify the contribution of the software in the context of the original research and to document the initial revision of the software in an archival sense.

The Machine Learning Toolbox (MaLT) software contains MATLAB scripts for running cross-validation experiments. A user guide for the code is presented in this appendix. Some details are intentionally omitted to focus attention on common usage. The interfaces and options for all important MaLT classes and functions are given in Appendix B.

## A.1 Getting Started

The Machine Learning Toolbox (MaLT) is a set of MATLAB classes and programs. MaLT is general enough to handle many learning scenarios; however, the current focus is kernel methods and multi-kernel classification.

Before we proceed to an example, it is necessary to have MATLAB (R14 or newer) installed. MATLAB is commercial software and requires a license. Academic licenses are available. In the following, we assume a LINUX operating system with `tcsh` shell.

The MaLT software should be unpacked into a directory, which we will call `MALTDIR`. Anywhere that `MALTDIR` appears in this document, it should be replaced with the correct path. Start MATLAB, and in a command window, enter "`addpath MALTDIR/src`" this will make the MaLT scripts easily available to MATLAB. Next, enter "`maltenv`" to set up the MaLT environment. At this point, all MaLT classes are ready for use.

## A.2 Example

The example experiment we will perform is a three-times replicated, five-fold cross-validation (3x5cv) over different settings of the $C$ parameter of a linear SVM, on a test data set consisting of 20 two-dimensional vectorial inputs with binary {-1,1} labels. This type of cross-validation over a parameter setting is an extremely common type of experiment. Of course, the example data set is small so the experiment will run quickly, and the results can be visualized. The example experiment script can easily be adapted for other purposes and provides a template for users.

The following example should provide enough information to get most users started with MaLT. The experiment is fully specified by the file `MALTDIR/results/examples/example1/example1.m`. The content of that file is repeated here in Figure A.1.

Let us look at the code in Figure A.1. The extremely compact size of the script indicates the convenience of MaLT as a platform for cross-validation experiments. Lines 4 and 5 load the test data set and labels into MATLAB matrices. Beginning on line 7, we create the cell array, `learners`, of MedmixLearner objects. Note the parameterization of the MedmixLearners. The learners vary in the $C$ soft margin parameter. This experiment will run each of the three learners over the same random splits of the data, so the results may be compared. We can, for example, empirically select a value for $C$ in this way. It is important to note that we can use any of the Learner classes with any parameter choices to construct an experiment. Beginning on line 12, we see a call to the function `malted`, the MaLT experiment driver. `malted` requires a Gram object representation of the data, a Labels object with class labels, a cell array of Learner objects, and an optional list of preferences.

Let us explore the `malted` parameters. On line 13, a Gram object is constructed for the test data. The Gram constructor is called with the result of the `compute` method of the default PolynomialKernel object along with the complete data matrix (twice). The Gram object will contain the dot product between all rows of the data matrix with itself. Therefore, the Gram object will contain a $20 \times 20$ Gram matrix. We could have used any of the available Kernel objects with any of their parameter settings. Alternatively, we can construct a Gram object from an existing Gram matrix in a file. We can also use more Gram matrices for multi-kernel learning.

The next `malted` parameter, on line 14, is the Labels object that provides the target labels used to train the learners. The Labels constructor is called in-place with the

```
%% Script to execute example cross-validation experiment
warning off all;
%% Load data and labels
load('../../../data/test/linear-data.txt');
load('../../../data/test/noisy-labels.txt');
%% Create Learner objects
learners={
  MedmixLearner({[1],[1]},'C',1);...
  MedmixLearner({[1],[1]},'C',10);...
  MedmixLearner({[1],[1]},'C',100)};
%% Call experiment driver
malted(
  Gram(compute(PolynomialKernel,linear_data,linear_data)),...
  Labels(noisy_labels),...
  learners,...
  'Folds',5,...
  'Seed',0);
```

Figure A.1: **The experiment script** for an example three-times replicated, five-fold, cross-validation experiment over $C = 1$, $C = 10$, and $C = 100$. The script sets up parameters and invokes the MaLT experiment driver program (`malted`) to do the real work.

matrix of class labels. The class label file could contain several columns, one for each target class we wish to learn. In the case of multiple classes, `malted` has several strategies for running an experiment. The default multi-class behavior is to run the experiment over the class labels in each column.

The final two `malted` parameters on lines 16 and 17 are part of the options list of *name, value* pairs. An option name precedes the desired value for the option. In our example, we choose to randomly partition the data into five folds, rather than the default two. We also specify a seed for the random number generator, to make the experiment deterministic (repeatable). Other options are available, as described in Section A.3.

Now, we will run the experiment. First, enter "`cd ../results/examples/example1/`" to move into the correct directory. Then, enter "`example1`" to execute the script. The various scripts and programs will generate diagnostic output that will scroll onto the command window. When this stops the program will indicate the outcome and elapsed time. The experiment should complete successfully.

The principal output of `malted` is a directory tree of results. The interesting files are at the leaf-level of the tree. The internal nodes segment the experiment according to several variables, over which `malted` iterates. Since `malted` is recoverable, and experiments are extensible, the directory tree always has the same structure. Re-starting an experiment or changing certain parameters is handled gracefully. When discussing the filesystem, the wildcard character (`*`) matches any sequence of characters.

We can test for results in several ways. In the current directory, the files "`gram.mat`" and "`labels.mat`" should have been created. These are MATLAB representations of objects containing the data and labels. A set of directories, with names of the form "`replication_*`" should have been created. Each of these should contain a

directory for each target class, for this example, "`class 1`." Also, there should be a file "`samples.mat`" that contains a random sampling of indices into the data. In each class directory, there should be a "`missing_0%`" directory and under that a directory for each of the five folds, "`fold_*`." In the fold directories are directories for each learning algorithm, in this example, "`Medmix Pos=1 Neg=1 C=*`." Finally, in the learner directories are the files "`state.mat`", "`train.mat`", and "`test.mat`." These leaf-level files contain the state of the trained learning algorithm and the classification result on training and testing data, respectively.

There is usually no need to verify the complete tree of results. In fact, it is not meant to be human readable. We view the result of an experiment with the `summarizeResults` script. In the current directory in the MATLAB command window, enter "`summarizeResults`" and the script will generate a table of scores for the different learners. In this example, we see one row of mean ROC AUC scores with one column for each setting of the regularization parameter, $C$. These scores can be compared to evaluate relative performance.

The statistical significance of the result can be computed with the `signedRank` script. Enter "`signedRank`" to compute a one-sided Wilcoxon signed-rank statistic for all pairs of learners in the experiment. The significance threshold is 0.05.

More example programs are included in the MaLT software distribution, along with other documentation. Those examples illustrate some of the more advanced MaLT features. The following sections of this appendix provide more detail regarding the scripts used in this example.

## A.3 malted

The MaLT experiment driver, `malted`, is used to run cross-validation experiments. In Section A.2, `malted` is called from the experiment script, `example1.m`, with appropriate arguments to run the 3x5cv experiment. Here, we discuss all of the arguments for `malted`.

**malted(*gram, labels, learners, optList*)**

*gram* - a Gram object or cell array of desired gram filenames. The files should be tab-delimited text with the first row containing column names and the first column containing row names. Other entries must be numeric.

*labels* - a Labels object or string containing the labels filename.

*learners* - a cell array of Learner objects

*optList* - a list of NAME,VALUE option pairs

***optList:***

'WorkDir' - directory for output files (default='.')

'Folds' - number of folds (default=2)

'Replications' - array of replications to run (default=[1,2,3])

'Classes' - cell array of class names (default=all columns in *labels*)

'Missing' - struct with percent missing, kernel indices, and type of affinity between missing examples (default=struct('percent', 0, 'kix', [], 'affinity', ''))

'Size' - size of random sample from data set (default=size of full data set)

'Mode' - one of {'OneVsAll', 'OneVsSample', 'Pairwise'} (default='OneVsAll')

'Seed' - random seed; given value or clock if value is [] (default=unchanged)

`malted` creates a hierarchy of directories containing the results of the specified k-fold cross-validation experiment. Refer to examples in the `MALTDIR/results/` directory for actual usage of various arguments.

## A.4 summarizeResults

The `summarizeResults` script computes aggregate results for `malted` experiments. A score type can be selected. The default is the area under the receiver operating characteristic curve (ROC AUC). A mean score is computed over the replications and folds for each learner, class, and percentile of missing data. The mean scores are printed in a table, and a struct array is returned with fields for the mean, standard error, and sample size for each table entry.

`summarizeResults` traverses a hierarchy of directories containing the results created by `malted`; it is very tolerant of incomplete results and attempts to return as much meaningful information as possible. The user can restrict the directories considered in the computation with optional parameters.

***stats* = summarizeResults(*optList*)**

*optList* - a list of NAME,VALUE option pairs

***optList:***

'WorkDir' - directory for work files (default='.')

'OutFile' - name for output file (default=stdout)

'Replications' - array of replications to run (default=[1;2;3])

'Classes' - cell array of classes to run {'class1';'class3'} (default=all classes)

'Percentiles' - array of percentiles of missing data to run: [10;50] (default=[0])

'Score' - type of score to compute: one of {'Tp', 'Tn', 'Fp', 'Fn', 'Accuracy', 'Sensitivity', 'Specificity', 'RocAuc'} (default='RocAuc')

'Format' - type of output: one of {'Tabular', 'HTML'} (default='Tabular')

'Compare' - append additional columns with difference between score columns. One column for each row in specifier: [1,3;1,4] (default=[])

## A.5 signedRank

The `signedRank` script computes the statistical significance of `malted` experiments. A score type can be selected. The default is the area under the receiver operating characteristic curve (ROC AUC). All scores for each learner, class, and percentile of missing data are used in the comparison. The Wilcoxon one-sided signed rank $p$-values are printed in a table, and a matrix is returned with the $p$-value for each table entry. The significance threshold is 0.05 and insignificant results are designated by "`---`" in the table.

`signedRank` traverses a hierarchy of directories containing the results created by `malted`; it is not tolerant of incomplete results. However, the user can restrict the directories considered in the computation with optional parameters.

***stats* = signedRank(*optList*)**

*optList* - a list of NAME,VALUE option pairs

***optList:***

'WorkDir' - directory for work files (default='.')

'OutFile' - name for output file (default=stdout)

'Replications' - array of replications to run (default=[1;2;3])

'Classes' - cell array of classes to run {'class1';'class3'} (default=all classes)

'Percentiles' - array of percentiles to run: [10;50] (default=[0])

'Score' - type of score to compute: one of {'Tp', 'Tn', 'Fp', 'Fn', 'Accuracy', 'Sensitivity', 'Specificity', 'RocAuc'} (default='RocAuc')

# Appendix B

# MaLT Reference

The Machine Learning Toolbox (MaLT) software contains MATLAB classes for creating and combining kernels, training classifiers, prediction, and cross-validation experiments. The code is described below in alphabetical order. For most users, Appendix A will be sufficient to demonstrate how to run cross-validation experiments. This appendix is provided primarily for those who wish to extend the MaLT implementation.

## B.1 Gram

Class for Gram matrix objects to be used by the learning algorithms. The Gram object contains $K$ Gram matrices, so it can be used in multi-kernel learning applications. A full Gram matrix must be symmetric, positive semidefinite and contain corresponding entries along rows and columns. Partial Gram matrices are often used for classification, e.g., one row for each test example and one column for each train example. The Gram object also handles this case.

### B.1.1 Gram (constructor)

Gram class constructor. Several variants are provided for different types of input arguments.

***gram* = Gram()**

***gram* = Gram(*gramObj, optList*)**

*gramObj* - Gram object

*optList* - list of NAME,VALUE option pairs

***gram* = Gram(*gramMatrices, optList*)**

*gramMatrices* - Matrix of $K$ $M \times N$ Gram matrices ($M \times N \times K$)

*optList* - list of NAME,VALUE option pairs

***gram* = Gram(*gramFiles, optList*)**

*gramFiles* - cell array of desired gram filenames. The files must be tab-delimited text with the first row containing column names and the first column containing row names. Other entries must be numeric.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Rows' - indices of rows to select

'Cols' - indices of columns to select

'Zero' - indices of rows/cols to zero

'Append' - A ($M \times N \times L$) matrix of Gram matrices to append to the Gram object. The new object will have $K + L$ matrices.

*gram*=Gram(compute(PolynomialKernel, $x$, $x$)) will construct a Gram object from the data matrix, $x$, using a linear kernel.

### B.1.2 get

Get class attributes.

***value* = get(*gramObj, name*)**

*gramObj* - Gram object

*name* - the name of the desired attribute

***name:***

'RowNames' - row names in cell array of strings

'ColNames' - column names in cell array of strings

### B.1.3 diag

Return the $K$ diagonals of the Gram object in a $(M \times K)$ matrix. The diagonals will be taken from the square matrix with corner at lower right and size equal to the number of rows. This is useful for the case in which the Gram matrices are not complete (square). This assumes the square test-test kernel entries are in the rightmost columns of the matrix. When the matrix is complete, this strategy also works.

***d* = diag(*gramObj*)**

*gramObj* - Gram object

### B.1.4 normalize

Normalize the gram matrices so the features lie on the surface of a unit hypersphere, $\hat{k}(x,z) = \frac{k(x,z)}{\sqrt{k(x,x)k(z,z)}}$. When the Gram matrices are not full, i.e., the diagonals do not contain all $k(x,x)$ or $k(z,z)$ entries, the diagonals must be provided in *optList*.

***gram* = normalize(*gramObj, optList*)**

*gramObj* - Gram object

*optList* - list of NAME,VALUE option pairs

***optList:***

'Diag1' - Diagonal entries associated with first dimension for $K$ Gram matrices in a $(M \times K)$ matrix.

'Diag2' - Diagonal entries associated with second dimension for $K$ Gram matrices in a $(N \times K)$ matrix.

### B.1.5 combine

Linearly combine the gram matrices. If no weights are given, an unweighted combination is computed.

***gram* = combine(*gramObj, optList*)**

*gramObj* - Gram object

*optList* - list of NAME,VALUE option pairs

***optList:***

'Weights' - Vector of $K$ weights for linear combination. To guarantee positive semidefiniteness, non-negative weights can be used. Alternatively, weights can be determined using SDP.

'Kernels' - A permutation of $1..K$ that indicates the mapping of weights to kernels.

## B.2 Kernel

### B.2.1 Kernel

A base class upon which classes for various kernel functions are implemented. Currently, there is no functionality in the base class. It only defines the inheritance hierarchy. The Gram object is used for Gram matrices and is a more general way to provide a kernel to a learning algorithm.

### B.2.2 PolynomialKernel

The family of polynomial kernels $k(x, z) = (ax^T z + b)^c$ is implemented by this class.

#### B.2.2.1 PolynomialKernel (constructor)

PolynomialKernel class constructor.

***poly* = PolynomialKernel()**

***poly* = PolynomialKernel(*polyObj*)**

*polyObj* - PolynomialKernel object

***poly* = PolynomialKernel(*optList*)**

*optList* - list of NAME,VALUE option pairs

***optList:***

'Coefficient' - the multiplicative constant, $a$ (default=1)

'Constant' - the additive constant, $b$ (default=0)

'Exponent' - the exponential constant, $c$ (default=1)

#### B.2.2.2 compute

Compute PolynomialKernel function value(s).

***k* = compute(*polyObj, u, v*)**

*polyObj* - object of class PolynomialKernel

*u* - row vector (or matrix) of test example(s)

*v* - row vector (or matrix) of train example(s)

### B.2.3 RadialKernel

The radial basis function kernels $k(x, z) = \exp(-\|x - z\|^2/\sigma^2)$ are implemented by this class.

#### B.2.3.1 RadialKernel (constructor)

RadialKernel class constructor.

***radial* = RadialKernel()**

***radial* = RadialKernel(*radialObj*)**

*radialObj* - RadialKernel object

***radial* = RadialKernel(*optList*)**

*optList* - list of NAME,VALUE option pairs

***optList:***

'Width' - the width constant, $\sigma$ (default=1)

#### B.2.3.2 compute

Compute RadialKernel function value(s).

***k* = compute(*radialObj, u, v*)**

*radialObj* - object of class RadialKernel

*u* - row vector (or matrix) of test example(s)

*v* - row vector (or matrix) of train example(s)

## B.3 Labels

A class to contain labels required by the supervised learning algorithms and cross-validation result objects. This is essentially a numeric matrix with cell arrays of strings for row and column names.

### B.3.1 Labels (constructor)

***labels* = Labels()**

***labels* = Labels(*labelsObj, optList*)**

*labelsObj* - object of class Labels

*optList* - list of NAME,VALUE option pairs

***labels* = Labels(*labelsMatrix, optList*)**

*labelsMatrix* - matrix of labels

*optList* - list of NAME,VALUE option pairs

***labels* = Labels(*labelsFile, optList*)**

*labelsFile* - name of desired labels file. The file must be tab-delimited text with the first row containing column names and the first column containing row names. Other entries must be numeric.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Rows' - indices of examples to select

'Cols' - indices of classes to select

'Classes' - names of classes to select

#### B.3.1.1 get

Get class attributes.

***value* = get(*labelsObj, name*)**

*labelsObj* - Labels object

*name* - the name of the desired attribute

***name:***

'Names' - cell array of row names

'ClassNames' - cell array of class (column) names

## B.4 Learned

### B.4.1 Learned

The base class for an object to hold the state of a trained classifier. Currently, there is no specific base class functionality. Learned serves as parent to the derived, learner-specific classes.

### B.4.2 MedmixLearned

A class to represent a trained MedmixLearner classifier. Objects of MedmixLearned can be used to classify new data. These objects also contain various statistics regarding the training, e.g. elapsed seconds.

#### B.4.2.1 MedmixLearned (constructor)

MedmixLearned class constructor. There is no need to directly call this function.

#### B.4.2.2 get

Get class attributes.

***value*** = **get(*medmixObj, name*)**

*medmixObj* - MedmixLearned object

*name* - the name of the desired attribute

***name:***

'labels' - train labels

'diag' - train Gram diagonal(s)

'observed' - train observed state

'elapsed' - elapsed seconds for training

'seed' - random seed for training

'best' - best re-init from training

'inits' - struct array of learned parameters for each re-init

#### B.4.2.3 classify

Predict class labels given a trained classifier and Gram object for a test data set. The result is returned in a Result object. Optionally, test labels can be provided for comparison. In this case, a CvResult object is returned.

***result* = classify(*medmixObj, gram, optList*)**

*medmixObj* - MedmixLearned object

*gram* - Gram object with test data

*optList* - list of NAME,VALUE option pairs

***optList:***

'Labels' - test Label object

'Missing' - struct of missing test data entries (default=[])

### B.4.3 MlmixLearned

A class to represent a trained MlmixLearner classifier. Objects of MlmixLearned can be used to classify new data. These objects also contain various statistics regarding the training, e.g. elapsed seconds.

#### B.4.3.1 MlmixLearned (constructor)

MlmixLearned class constructor. There is no need to directly call this function.

#### B.4.3.2 get

Get class attributes.

***value* = get(*mlmixObj, name*)**
*mlmixObj* - MlmixLearned object
*name* - the name of the desired attribute

***name:***
'elapsed' - elapsed seconds for training
'seed' - random seed for training
'best' - best re-init from training
'inits' - struct array of learned parameters for each re-init

#### B.4.3.3 classify

Predict class labels given a trained classifier and Gram object for a test data set. The result is returned in a Result object. Optionally, test labels can be provided for comparison. In this case, a CvResult object is returned.

***result* = classify(*mlmixObj, gram, optList*)**
*mlmixObj* - MlmixLearned object
*gram* - Gram object with test data
*optList* - list of NAME,VALUE option pairs

***optList:***

'Labels' - test Label object

### B.4.4 SdpLearned

A class to represent a trained SdpLearner classifier. Objects of SdpLearned can be used to classify new data. These objects also contain various statistics regarding the training, e.g. elapsed seconds.

#### B.4.4.1 SdpLearned (constructor)

SdpLearned class constructor. There is no need to directly call this function.

#### B.4.4.2 get

Get class attributes.

***value* = get(*sdpObj*, *name*)**

*sdpObj* - SdpLearned object

*name* - the name of the desired attribute

***name:***

'elapsed' - elapsed seconds for training

'labels' - train labels

'diag' - train Gram diagonal(s)

'muOpt' - the kernel combination weights

#### B.4.4.3 classify

Predict class labels given a trained classifier and Gram object for a test data set. The result is returned in a Result object. Optionally, test labels can be provided for comparison. In this case, a CvResult object is returned.

***result* = classify(*sdpObj, gram, optList*)**

*sdpObj* - SdpLearned object

*gram* - Gram object with test data

*optList* - list of NAME,VALUE option pairs

***optList:***

'Labels' - test Label object

## B.5 Learner

### B.5.1 Learner

A base class from which all classes for learning algorithms are derived. At this time, all derived learners are supervised algorithms. However, the Learner interface is completely generic and can support unsupervised techniques, as well.

### B.5.2 MedmixLearner

This class implements the maximum entropy discrimination (MED) mixture of Gaussians classifier. The discriminant is based on the log ratio of two Gaussian mixture models. The technique permits each Gaussian component to reside in a separate

feature space, thus allowing a combination of kernels. This class requires one of the optimizers, listed below, to be installed.

#### B.5.2.1 MedmixLearner (constructor)

MedmixLearner class constructor.

***medmix* = MedmixLearner()**

***medmix* = MedmixLearner(*medmixObj*)**

*medmixObj* - MedmixLearner object

***medmix* = MedmixLearner(*components, optList*)**

*components* - cell array of positive class and negative class mixture components. In this example, $\{[1,3],[1]\}$, the positive class mixture has two components that use kernels 1 and 3 from the GramObject and the negative class mixture has one component and uses only kernel 1.

*optList* - list of NAME,VALUE option pairs

***optList:***

'C' - regularization parameter (default=100)

'Thresh' - convergence threshold (default=1e-3)

'MaxIter' - maximum iterations (default=50)

'NumInits' - number of re-inits (default=1)

'Optimizer' - 'QUADPROG', 'MOSEK', 'SMO' (default='QUADPROG')

'Observe' - observe mixture state (default=[])

#### B.5.2.2 get

Get class attributes.

***value* = get(*medmixObj, name*)**

*medmixObj* - MedmixLearner object

*name* - the name of the desired attribute

***name:***

'Components' - the component-to-kernel mappings

'C' - regularization parameter

'Thresh' - convergence threshold

'MaxIter' - maximum iterations

'Optimizer' - 'QUADPROG', 'MOSEK', 'SVLAB', 'SMO'

'Observe' - observed mixture state

'Uname' - uniquely derived name based on parameterization. This is useful in creating directory names for experimental results.

#### B.5.2.3 train

Train a MED Gaussian mixture classifier given a MedmixLearner object, training examples and labels, and a list of options. The returned trained classifier can be used to predict labels for test examples.

***learned* = train(*medmixObj, gramObj, labelsObj, optList*)**

*medmixObj* - MedmixLearner object

*gramObj* - training Gram object. Can include test data in last rows/cols for transductive applications.

*labelsObj* - training Labels object. Must contain corresponding train labels only.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Seed' - value for random number generator seed. Use [] for clock. (default=unchanged)

'Missing' - struct of missing data entries (default=[])

'Debug' - enforce strict bug checking (default=true)

### B.5.3 MlmixLearner

This class implements the maximum likelihood (ML) mixture of Gaussians classifier. The discriminant is based on the log ratio of two Gaussian mixture models. The technique permits each Gaussian component to reside in a separate feature space, thus allowing a combination of kernels.

#### B.5.3.1 MlmixLearner (constructor)

MlmixLearner class constructor.

***mlmix* = MlmixLearner()**

***mlmix* = MlmixLearner(*mlmixObj*)**

*mlmixObj* - MlmixLearner object

***mlmix* = MlmixLearner(*components, optList*)**

*components* - cell array of positive class and negative class mixture components. In this example, $\{[1,3],[1]\}$, the positive class mixture has two components that use kernels 1 and 3 from the GramObject and the negative class mixture has one component and uses only kernel 1.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Thresh' - convergence threshold (default=1e-2)

'MaxIter' - maximum iterations (default=50)

'NumInits' - number of re-inits (default=1)

#### B.5.3.2 get

Get class attributes.

***value* = get(*mlmixObj, name*)**

*mlmixObj* - MlmixLearner object

*name* - the name of the desired attribute

***name:***

'Components' - the component-to-kernel mappings

'Thresh' - convergence threshold

'MaxIter' - maximum iterations

'Uname' - uniquely derived name based on parameterization. This is useful in creating directory names for experimental results.

#### B.5.3.3 train

Train a ML Gaussian mixture classifier given a MlmixLearner object, training examples and labels, and a list of options. The returned trained classifier can be used to predict labels for test examples.

***learned*** = **train(*mlmixObj, gramObj, labelsObj, optList*)**

*mlmixObj* - MlmixLearner object

*gramObj* - training Gram object. Can include test data in last rows/cols for transductive applications.

*labelsObj* - training Labels object. Must contain corresponding train labels only.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Seed' - value for random number generator seed. Use [] for clock. (default=unchanged)

'Debug' - enforce strict bug checking (default=true)

### B.5.4 SdpLearner

This class implements the SDP combination of kernels within a support vector machine (SVM) classifier. The discriminant is the SVM with a linear combination of kernels.

### B.5.4.1 SdpLearner (constructor)

SdpLearner class constructor. At present, this class requires the Mosek optimization software, which is not free.

***sdp* = SdpLearner()**

***sdp* = SdpLearner(*sdpObj*)**

*sdpObj* - SdpLearner object

***sdp* = SdpLearner(*kernels, optList*)**

*kernels* - array of kernels to use. In this example, $[1, 3]$, the kernels 1 and 3 from the GramObject are combined.

*optList* - list of NAME,VALUE option pairs

***optList:***

'C' - regularization parameter (default=100)

'Mode' - optimization mode for kernel weights. 'Opt' is SDP optimized. 'NoOpt' is equal weight. (default='Opt')

'Weights' - vector of fixed kernel combination weights (default=[])

'Normalize' - re-normalize kernels after combination (default=false)

### B.5.4.2 get

Get class attributes.

***value* = get(*sdpObj, name*)**

*sdpObj* - SdpLearner object

*name* - the name of the desired attribute

***name:***

'Kernels' - the kernel indices

'C' - regularization parameter

'Mode' - optimization mode for kernel weights. 'Opt' is SDP optimized. 'NoOpt' is equal weight.

'Weights' - vector of kernel combination weights

'Normalize' - re-normalize kernels after combination

'Uname' - uniquely derived name based on parameterization. This is useful in creating directory names for experimental results.

#### B.5.4.3 train

Train a multi-kernel SDP SVM classifier given a SdpLearner object, training examples and labels, and a list of options. The returned trained classifier can be used to predict labels for test examples.

***learned* = train(*sdpObj, gramObj, labelsObj, optList*)**

*sdpObj* - SdpLearner object

*gramObj* - training Gram object. Can include test data in last rows/cols for transductive applications.

*labelsObj* - training Labels object. Must contain corresponding train labels only.

*optList* - list of NAME,VALUE option pairs

***optList:***

'Debug' - enforce strict bug checking (default=true)

# B.6 Result

## B.6.1 Result

A class for the result of a classification experiment. The Result object contains the predictions and discriminant values.

### B.6.1.1 Result (constructor)

Constructor for Result.

***result* = Result()**

***result* = Result(*resultObj*)**

*resultObj* - Result object

***result* = Result(*disc, pred, ep, en*)**

*disc* - column vector of discriminant values

*pred* - column vector of predicted labels

*ep* - matrix of expected log likelihood for positive model or []

*en* - matrix of expected log likelihood for negative model or []

### B.6.1.2 get

Get class attributes.

***value* = get(*resultObj, name*)**

*resultObj* - Result object

*name* - the name of the desired attribute

***name:***

'Disc' - column vector of discriminant values

'Pred' - column vector of predicted labels

'Ep' - matrix of expected log likelihood for positive model, if applicable.

'En' - matrix of expected log likelihood for negative model, if applicable.

## B.6.2 CvResult

CvResult extends the Result class for cross-validation experiments, where the test labels are known. The CvResult class allows the computation of various scores of classifier performance. The scores are retrieved with the `get` method.

### B.6.2.1 CvResult (constructor)

Constructor for CvResult.

***cvresult* = CvResult()**

***cvresult* = CvResult(*cvresultObj*)**

*cvresultObj* - CvResult object

***cvresult* = CvResult(*resultObj, labels*)**

*resultObj* - Result object

*labels* - Label object with true class labels

### B.6.2.2 get

Get class attributes.

***value* = get(*cvresultObj, name*)**

*cvresultObj* - CvResult object

*name* - the name of the desired attribute

***name:***

'Disc' - column vector of discriminant values

'Pred' - column vector of predicted labels

'Ep' - matrix of expected log likelihood for positive model, if applicable.

'En' - matrix of expected log likelihood for negative model, if applicable.

'Tp' - number of true positive predictions

'Tn' - number of true negative predictions

'Fp' - number of false positive predictions

'Fn' - number of false negative predictions

'Accuracy' - accuracy

'Sensitivity' - sensitivity

'Specificity' - specificity

'RocAuc' - area under the receiver operating characteristic

## B.7 smo

SMO is sequential minimal optimization, an iterative algorithm for solving an optimization problem with analytic steps over a minimal number of variables. We provide an implementation of SMO for the MedmixLearner. The code is written in C++ and must be compiled for the hardware/operating system platform. MATLAB includes a program called `mex` to compile C code for use with the MATLAB MEX interface. The

`mex` documentation is the definitive reference; however, it should be sufficient to `cd MALTDIR/src/smo` and edit the `mexopts.sh` file, as necessary for the desired platform. Then the command "`mex smo.cpp`" will build the binary code that MATLAB needs to run SMO.

The final step is to specify that MedmixLearner should use SMO. That is done by providing the optional parameters "`'Optimizer', 'SMO'`" to the MedmixLearner constructor.