University of Alberta

Multiple Kernel Learning with Many Kernels

by

Arash Afkanpour

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Abstract

Multiple kernel learning (MKL) addresses the problem of learning the kernel function from data. Since a kernel function is associated with an underlying feature space, MKL can be considered as a systematic approach to feature selection. Many of the existing MKL algorithms perform kernel learning by combining a given set of base kernels. While efficient MKL algorithms are capable of handling large training sets, their computational complexity depends linearly on the number of base kernels. Hence, these algorithms are not scalable to problems with many kernels.

In this thesis, we investigate MKL when the number of kernels is very large. We aim to exploit the structure of kernel space to design efficient MKL algorithms. Such a structure exists for some of the important families of kernels, such as polynomial kernels and Gaussian kernels. We propose efficient gradient-based algorithms, with convergence guarantees, for the two prominent problem formulations of MKL, i.e. one-stage and two-stage MKL. We show that stochastic gradient descent and greedy coordinate descent are suitable algorithms to deal with very large kernel sets.

We propose an efficient stochastic gradient descent method for one-stage MKL. We show that sampling a coordinate with a probability proportional to the magnitude of gradient results in a low-variance estimate of gradient. For the case of learning polynomial kernels we show that the computational complexity of our algorithm has only a logarithmic dependence on the number of kernels. We show how greedy coordinate descent can be applied to one-stage and two-stage MKL. Greedy coordinate descent is in particular useful when the goal is to combine continuously-parameterized kernels, such as Gaussian kernels.

We perform thorough empirical study on synthetic and real data to compare the new gradient-based algorithms against several MKL algorithms. Our experiments demonstrate that our new methods are competitive in terms of generalization performance, while their computational cost is significantly less than other methods that enjoy similarly good generalization performance.

Acknowledgements

Completing the PhD program is probably the most challenging task in my life so far. There are many people who helped me during this journey and I am grateful to all of them.

I would like to thank my supervisors, Csaba Szepesvári and Michael Bowling for their help and support.

I learned a great deal from Csaba during these years. He taught me how mathematics can be properly used to formulate, analyze and solve machine learning problems. I learned from him how to write clearly and accurately. Csaba has always been accessible and willing to help even at times that he was swamped with deadlines, courses, and meetings and I am grateful for that. Thanks Csaba!

Mike Bowling has always been a sharp and talented supervisor who shows a great deal of passion in doing research. I am deeply thankful to him for his guidance and support from the very beginning of my PhD studies.

I would like to thank my co-author and friend András György for his collaboration. We have had many insightful discussions together and I learned a lot from him.

I would like to thank Dale Schuurmans, Ivan Mizera, Nilanjan Ray, and Massimiliano Pontil for being on my thesis committee and for their valuable questions and constructive feedback on my research.

I would like to thank my friends at the University of Alberta. In particular I would like to thank Amir-massoud Farahmand and Saman Vaisipour. I remember countless number of times that Amir-massoud and I were discussing interesting topics and problems, whether related to our research or not, and how much I learned from these discussions. Saman has been my best friend since we started B.Sc. in 2000. He is the kind of friend who is always there for you when you need one.

I would like to thank my parents, Ali and Mahnaz, for their unconditional love and support throughout my life. They always provided me with an environment in which I could pursue my dreams and achieve my goals. Thank you both.

Most of all, I would like to thank my wonderful wife, Fariba Mahdavifard, for her support, patience, and encouragement. This path was not an easy one and there were many times that I felt hopeless and frustrated. She has always listened to my complaints and helped me overcome obstacles. I am very thankful to her.

This work was supported financially by the Alberta Innovates Technology Futures.

Table of Contents

Chapter 1

Introduction

In machine learning it is well-known that the choice of features heavily influences the performance of learning methods. Flexible algorithms that can perform feature selection automatically while learning a predictor would be highly desirable. Similarly, the performance of a learning method that uses a kernel function is highly dependent on the choice of kernel. In this context, feature selection translates into using multiple kernels and allowing the learning algorithm to choose the best kernel (combination).

Multiple kernel learning (MKL) provides a systematic approach to using data to learn the most suitable kernel function for a learning task. These methods are usually designed to combine kernels from a given set. Since a kernel function corresponds to a notion of similarity between data instances, combining multiple kernels can be interpreted as combining different notions of similarity. The similarity derived from a kernel is the inner product in some underlying feature space. Hence, multiple kernel learning corresponds to feature selection. Prior to work in multiple kernel learning, choosing the best kernel (and tuning the corresponding kernel parameters) was performed by cross validation. Multiple kernel learning, however, has been proposed as an alternative for learning the best kernel combination and predictor at the same time.

The reason that multiple kernel learning is popular is because often it is hard to decide *a priori* which of a number of features-maps (or kernels) is the most appropriate for a given task. In fact, in many learning problems one has very little a priori information that helps this choice. Hence, letting the learning algorithm find the best combination of a set of kernels is an attractive idea. For some of the important kernel families, such as polynomial kernels or Gaussian kernels, the number of base kernels could be very large. While there are heuristic methods to decrease the number of base kernels, it is desirable to directly feed a large set of kernels to an algorithm and let it find the best combination.

There exists a large body of work in the literature of MKL. The main challenge that has been addressed by many of the "efficient" MKL algorithms is learning with large datasets. Much less attention, however, has been paid to the other dimension of scalability, i.e. handling large number of kernels. The computational complexity of most of the MKL algorithms depends linearly on the number of base kernels. Hence, these algorithms are not scalable to problems with many kernels.

In this thesis, we investigate the computational challenge of multiple kernel learning when the number of kernels is very large. In particular, we are interested in exponentially or infinitely large kernel sets. Examples include sets of base kernels that come from a space with combinatorial structure and thus their number could be exponentially large, such as polynomial kernels; and uncountable sets of parameterized base kernels whose parameters belong to a continuous set, such as Gaussian kernels.

There are two well-known problem formulations of MKL, i.e. one-stage and two-stage MKL. In one-stage MKL, following the structural risk minimization framework, the optimization problem is formulated to learn the kernel weights and the prediction function in one stage. In two-stage MKL, on the other hand, the learning problem is divided into two separate stages. In the first stage the kernel combination is learned using a surrogate loss function that measures the quality of a kernel function. The learned kernel function is then used in the second stage to learn a predictor using standard kernel algorithms such as support vector machines (SVM) or kernel ridge regression. For both approaches, we propose efficient algorithms based on stochastic gradient descent and greedy coordinate descent.

For one-stage MKL, like some previous work (e.g. Rakotomamonjy et al., 2008; Xu et al., 2008; Kloft et al., 2011) we start with the approach that views the MKL problem as a two-step iterative procedure where the first step optimizes the weights of the kernels to be combined, and the second step computes the predictor weights. More specifically, we aim to minimize the group p-norm penalized empirical risk. However, as opposed to these algorithms whose per iteration complexity depends linearly on the number of kernels, following (Nesterov, 2010, 2012; Shalev-Shwartz and Tewari, 2011; Richtárik and Takáĉ, 2011) we use a randomized coordinate descent method that improves per iteration complexity. In particular, we show that in the case of learning polynomial kernels, the computational complexity of our algorithm has a logarithmic dependence on the number of kernels. In our method, randomization is used to build an unbiased estimate of the gradient at the most recent iteration. The issue then is how the variance (and so the number of iterations required) scales with the number of kernels. To address this issue, we propose to make the distribution over the updated coordinate dependent on the history. We will argue that sampling from a distribution that is proportional to the magnitude of the gradient vector is desirable to keep the variance low and in fact we will show that there are interesting cases of MKL (in particular, learning polynomial kernels) when efficient sampling (i.e., sampling at a logarithmic cost) is feasible from this distribution. Then, the variance is controlled by the a priori weights put on the kernels, making it potentially independent of the number of kernels. We show that under these favorable conditions (and in particular, for the polynomial kernel set with some specific prior weights), the complexity of the method depends logarithmically on the number of kernels, which makes our MKL algorithm efficient even for an exponential number of kernels. This is to be contrasted to the approach of Nesterov (2010, 2012) where a fixed distribution is used and where the a priori bounds on the method's convergence rate, and hence, its computational cost to achieve a prescribed precision, will depend linearly on the number of kernels.[1] Our algorithm is based on the mirror descent algorithm (similar to the work of Richtárik and Takáĉ (2011) who uses uniform distributions).

For two-stage MKL, we propose a greedy coordinate descent algorithm to optimize the *centered kernel alignment* metric of Cortes et al. (2010), which was originated from the uncentered alignment metric of Cristianini et al. (2002). We show that our algorithm also fits in the framework of Forward Stagewise Additive Modeling (FSAM), which is related to boosting (see, e.g. Hastie et al., 2001, Chapter 10). Our new greedy coordinate descent algorithm for MKL is particularly suited to learning a combination of base kernels from a parameterized family of kernels, $(\kappa_\sigma)_{\sigma\in\Sigma}$, where Σ is a "continuous" parameter space, i.e., some subset of a Euclidean space. A prime example and popular choice is when κ_σ is a Gaussian kernel, where σ can be a single common bandwidth or a vector of bandwidths, one per coordinate. One approach to apply MKL with such continuously-parameterized base

[1]Note that we are comparing upper bounds here, so the actual complexity could be smaller.

kernels, which is adopted by many practitioners, is to discretize the parameter space Σ into r values and then find an appropriate combination of the resulting set of base kernels, $\mathcal{S} = \{\kappa_{\sigma_1}, \ldots, \kappa_{\sigma_r}\}$. The advantage of this approach is that once the set $\mathcal{S}$ is fixed, any of the standard MKL methods available in the literature can be used to find the coefficients for combining the base kernels in $\mathcal{S}$ (see the papers by Lanckriet et al. 2004; Sonnenburg et al. 2006; Rakotomamonjy et al. 2008; Cortes et al. 2009a; Kloft et al. 2011 and the references therein). One potential drawback of this approach, however, is that it requires an appropriate, a priori choice of $\mathcal{S}$. This might be problematic, e.g., if Σ is contained in a Euclidean space of moderate, or large dimension (e.g., dimension over 20) since the size of $\mathcal{S}$ grows exponentially with dimensionality. Furthermore, independent of the dimensionality of the parameter space, the need to choose the set $\mathcal{S}$ independently of the data is at best inconvenient and selecting an appropriate resolution might be far from trivial. We, therefore, propose an alternative method which avoids the need for discretizing the space Σ.

We are not the first to realize that discretizing a continuous parameter space might be troublesome: The method of Argyriou et al. (2005), and later Argyriou et al. (2006), can also work with continuously parameterized spaces of kernels. We shall see in Chapter 5 that the algorithm of Argyriou et al. (2005) is indeed an instance of greedy coordinate descent. While our algorithm and the method of Argyriou et al. (2005) are both greedy coordinate descent algorithms, we aim to solve the two-stage MKL problem, while their algorithm solves one-stage MKL.

The greedy selection of coordinates in the context of MKL translates into solving an optimization problem over $\sigma \in \Sigma$ with an objective function that is a linear function of the kernel matrix associated with the kernel function κ_σ. Because of the non-linear and non-convex dependence of this matrix on σ, this step must resort to local optimization. However, as we shall demonstrate empirically, even if we use local solvers to solve this optimization step, the algorithm still shows an overall excellent performance. Similar results were also obtained by Argyriou et al. (2005). This is not completely unexpected considering the connection between greedy coordinate descent and boosting, and that one of the key ideas underlying boosting is that it is designed to be robust even when the individual "greedy" steps are imperfect (cf., Chapter 12, Bühlmann and van de Geer 2011).

1.1 Contributions

In this thesis, we propose a novel approach to MKL based on gradient descent that is able to select and combine kernels from very large sets. The approach is shown to be applicable to both one-stage and two-stage MKL. In particular, it is shown that in both cases the risk functions can be chosen to be convex without compromising the algorithm's ability to deal with many kernels. We show that stochastic gradient descent and greedy coordinate descent are suitable algorithms to deal with very large kernel sets. The following list provides an overview of contributions of this thesis:

- We propose a new stochastic gradient descent algorithm for high-dimensional optimization. We show that by letting the sampling distribution, and hence, the gradient estimate be dependent on history in some high-dimensional problems an exponential boost of the rate of convergence of the algorithm is possible. Finite-time convergence bounds are given that make the dependence on the chosen distribution explicit. As the bounds do not directly depend on the number of dimensions of the decision variable, if an appropriate sampling procedure can be designed, the algorithm can be very efficient for high-dimensional optimization.

- We show how our new stochastic gradient descent algorithm can be applied to the problem of one-stage MKL. We investigate the conditions required that guarantee the convergence of the algorithms in the context of MKL. We investigate the important problem of learning polynomial kernels and show how the variance of the gradient estimate can be controlled in this problem. This leads to an efficient and scalable MKL algorithm for learning polynomial kernels whose computational complexity is only polynomial in the number of base kernels, as opposed to previous methods that all scale exponentially as a function of this number.

- We show how greedy coordinate descent can be applied to one-stage and two-stage MKL. Greedy coordinate descent is in particular useful when the goal is to select and combine kernels from a parameterized set with infinitely-many kernels. While the algorithm of Argyriou et al. (2005) is an instance of greedy coordinate descent for one-stage MKL, we show how this method can be applied to two-stage MKL too. We show that the existing convergence results for greedy coordinate descent guarantees that for two-stage MKL this algorithm converges even if it is applied to kernel sets with infinitely-many kernels.

- We perform thorough experiments on synthetic and real data to compare the new gradient-based algorithms against several MKL algorithms. Our experiments serve multiple purposes.

  We show that sampling from a distribution that is proportional to the magnitude of the gradient is beneficial. We, therefore, use this sampling procedure in our stochastic gradient descent algorithm to learn polynomial kernels. Our experiments also demonstrate that our new methods are competitive in terms of generalization performance, while their computational cost is significantly less than that of the other methods that enjoy similarly good generalization performance.

  We then show how this algorithm can be applied to combine kernels from a set with infinitely-many kernels. In particular, we show the application of our method to learn a combination of Gaussian kernels.

  Next, we experiment with our greedy coordinate descent algorithm in one-stage and two-stage MKL. We explore the potential advantages, as well as limitations of the proposed technique. While the greedy kernel selection procedure needs to solve a non-convex problem for a wide range of kernel families, including Gaussian kernels, we show that the algorithm is indeed reliable. We show that this method can be successfully applied even when Σ is a subset of a multi-dimensional space.

  Our experiments demonstrate a clear advantage of greedy coordinate descent algorithms that avoid discretization of the kernel parameter space. This conclusion is strengthened by the fact that in the experiments the algorithm of Argyriou et al. (2005), which can be considered as greedy coordinate descent for one-stage MKL and our greedy coordinate descent algorithm for two-stage MKL perform better than methods that require discretization of kernel parameter space. While using finer discretizations of the parameter space and providing more base kernels may improve performance of finite MKL methods, the additional computational complexity makes this approach impractical for high dimensional parameterizations. A greedy coordinate descent MKL algorithm, however, can still produce meaningful kernel combinations in this case.

Apart from the main contributions stated above, we also present several other results:

- We show that for two-stage MKL, maximizing alignment can be equivalently solved by minimizing the Euclidean distance between the kernel combination and the ideal kernel. While Cortes et al. (2010) had previously shown that alignment maximization can be done by means of a quadratic programming problem, to the best of our knowledge the relation between alignment and Euclidean distance has not been reported in the literature.

- We show empirically that combining multiple kernels achieves better generalization error compared to picking one kernel from a set.

- Finally, our experiments reveal an interesting novel insight into the behavior of two-stage methods: we noticed that two-stage methods can "overfit" the performance metric of the first stage. In some problems we observed that our method could find kernels that gave rise to better (test-set) performance on the first-stage metric, while the method's overall performance degrades when compared to using kernel combinations whose performance on the first metric is worse. The metric of the first stage is only a surrogate performance measure and thus better performance according to this surrogate metric does not necessarily result in better performance in the primary metric. Furthermore, we actually see this effect in practice. However, we show that with proper capacity control, the problem of overfitting the surrogate metric can be overcome. To the best of our knowledge, this phenomenon has not been reported in the past for two-stage algorithms.

The thesis is organized as follows: In Chapter 2, we review background material, including the problem formulation of multiple kernel learning. In this chapter we also review some of the previous MKL algorithms related to our work. Chapter 3 provides examples to demonstrate the importance of MKL as opposed to using a single kernel. In this chapter we also discuss the importance of designing algorithms that go beyond current approaches in MKL. We then propose new stochastic gradient descent (Chapter 4) and greedy coordinate descent (Chapter 5) algorithms for

MKL with many kernels. Experimental results are presented in Chapter 6. Finally, in Chapter 7 we give some final remarks and propose further directions to extend this work.

Chapter 2

Background and Related Work

The goal of machine learning is to develop algorithms that can find patterns in data. In this thesis, we focus on *supervised learning.* In supervised learning, we are given a set of *training* examples

$$\{(x_1, y_1), \ldots, (x_n, y_n)\},$$

where $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The input space usually has a vector form, i.e. $\mathcal{X} \subseteq \mathbb{R}^d$, however, non-vector types can be considered too. Problems in which the output values come from a finite set are called *classification* problems. In *binary classification* output values belong to the set $\mathcal{Y} = \{-1, 1\}$. One example of a classification problem is handwritten digit recognition in which the training data consists of images of handwritten digits along with a label for each image that specifies what digit the image represents. A supervised learning algorithm in this example can be used to build a *predictor* that assigns a digit label to any input image. On the other hand, *regression* problems are problems in which the output values are real numbers. For example, the problem of predicting the amount of rainfall is a regression problem. In general, given the training data, the goal is to build a prediction function f that maps the input space $\mathcal{X}$ to the output space $\mathcal{Y}$. Learning a predictor from training data is based on the assumption that data are generated, usually in an *independent and identically distributed* (i.i.d.) manner, from a common underlying distribution $P(X, Y)$. Note that this assumption is necessary to build a predictor with high generalization performance. Prediction function f is the minimizer of a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$, over the input-output space:

$$\min_{f \in \mathcal{F}} \quad \int_{\mathcal{X} \times \mathcal{Y}} \ell(y, f(x))\, dP(x, y),$$

where P is the distribution of data, and $\mathcal{F}$ is a class of functions. The distribution of data P is usually not known. One approach is to minimize an *empirical estimate* of the actual loss function which is obtained over training data,

$$\min_{f \in \mathcal{F}} \quad \sum_{i=1}^{n} \ell(y_i, f(x_i)) + \lambda \Omega(f).$$

Since the class of functions $\mathcal{F}$ is typically very large and the size of training data is (relatively) small, one needs to control the complexity of f to avoid overfitting. This is done by adding a *regularization term* $\Omega(f)$ to the minimization problem that gives a large penalty to complex functions. Here, $\lambda > 0$ is the *regularization coefficient* that controls the tradeoff between empirical accuracy and the complexity of f. Without the regularization term, it is very likely that the predictor will overfit the training data and would not generalize well to unseen data.

2.1 Kernel Methods

Kernel methods are a family of algorithms for building linear models in a high-dimensional feature space (Cortes and Vapnik, 1995). The predictor built by a kernel method has the following form:

$$f(x) = \langle w, \phi(x) \rangle,$$

where $w \in \mathcal{W}$ is the corresponding weight vector and $\phi : \mathcal{X} \to \mathcal{W}$ maps each instance onto feature space $\mathcal{W}$. Feature space $\mathcal{W}$ could be high- or even infinite-dimensional. An important property of kernel methods, which makes it possible to deal with a high-dimensional feature space, is that it is not necessary to explicitly compute $\phi(x)$. The only information that a kernel algorithm needs is the inner product of pair of instances in feature space. This inner product is defined through a *kernel function*:

$$\kappa(x, x') = \left\langle \phi(x), \phi(x') \right\rangle.$$

A kernel function, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, is a positive semidefinite function. That is, κ is a symmetric function and the resulting matrix of applying κ to any finite subset of $\mathcal{X}$ is positive semidefinite.

For a given kernel function κ, consider the following function space

$$\mathcal{H}_0 = \left\{ \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot) : n \in \mathbb{N},\, x_i \in \mathcal{X},\, \alpha_i \in \mathbb{R},\, i = 1, \ldots, n \right\}.$$

For any two functions $f, g \in \mathcal{H}_0$ given by

$$\begin{aligned} f &= \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot), \quad \text{and} \\ g &= \sum_{j=1}^{m} \beta_j \kappa(x_j, \cdot), \end{aligned}$$

the inner product of f and g is defined as

$$\langle f, g \rangle \;=\; \sum_{i=1}^{n} \sum_{j=1}^{m} \alpha_i \beta_j \kappa(x_i, x_j). \tag{2.1}$$

The *Reproducing Kernel Hilbert Space* (RKHS) underlying κ is defined as the closure of $\mathcal{H}_0$ with respect to the norm induced by this so-defined inner product. This will be denoted by $\mathcal{H}$. Note that if $f = \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot) \in \mathcal{H}$ then

$$\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j \kappa(x_i, x_j) \geq 0, \tag{2.2}$$

in which $\|f\|_{\mathcal{H}}$ is the norm of f induced by the inner product underlying κ. This property holds since the kernel function, κ, is positive semidefinite. From (2.1) and (2.2) it is clear that $\langle f, g \rangle$ is a real number. Furthermore, $\langle \cdot, \cdot \rangle$ is symmetric and bilinear. Therefore, it satisfies the properties of an inner product.

Assuming that f has the above mentioned expansion, it also follows from (2.1) that

$$\langle f, \kappa(x, \cdot) \rangle = \sum_{i=1}^{n} \alpha_i \kappa(x_i, x) = f(x),$$

and for any $x, x' \in \mathcal{X}$

$$\big\langle \kappa(x, \cdot), \kappa(x', \cdot) \big\rangle = \kappa(x, x').$$

This is called the *reproducing property* of kernel k.

The *representer theorem* (Kimeldorf and Wahba, 1971), which is the heart of kernel methods, states that the solution of a large class of problems can be expressed as a kernel expansion over training data points. Let $\ell : \mathbb{R} \times \mathbb{R} \to [0, \infty)$ be an arbitrary loss function. Any function $f \in \mathcal{H}$ that minimizes

$$\sum_{i=1}^{n} \ell(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2, \tag{2.3}$$

has the following representation:

$$f = \sum_{i=1}^{n} \alpha_i \kappa(x_i, \cdot).$$

This key property enables learning f through learning $\alpha = (\alpha_i)_{i=1}^n$. In other words, the computational complexity of learning f depends only on the number of data points and is independent of the number of dimensions of the feature space underlying κ. It is usually assumed that the loss function ℓ is convex in its second argument in order to guarantee the convexity of the functional (2.3). Many of the loss functions currently used in machine learning such as the squared-loss, $\ell(y, z) = \frac{1}{2}(y-z)^2$, and the hinge-loss, $\ell(y, z) = \max(1 - yz, 0)$, satisfy this condition. For regression problems, using the squared-loss gives the *kernel ridge regression* algorithm. For classification problems, using the hinge-loss results in the well-known soft margin *support vector machine* (SVM) algorithm.

Another appealing characteristic of kernel methods is that, unlike many other learning algorithms, these methods are independent of input type. That is they are applicable to non-vector input types such as graphs, strings, etc. Once we have a valid kernel function that is applicable to the instances of the problem at hand, any standard kernel algorithm can be used to learn a predictor. Further details of kernel methods can be found in, e.g., Schölkopf and Smola (2002); Shawe-Taylor and Cristianini (2004); Steinwart and Christmann (2008).

The kernel function determines the underlying RKHS. Hence, given a limited amount of training data, choosing the right kernel function for an application is an important part of learning a predictor with high accuracy. The traditional approach to choosing the kernel function is to select the best kernel function among a set of candidates, $\{\kappa_1, \ldots, \kappa_r\}$, according to their prediction accuracy on a validation set. More recently, multiple kernel learning (MKL) provides a flexible approach by learning a kernel function that is a combination of this kernel set.

2.2 Problem Formulation of Multiple Kernel Learning

In this section we present the problem formulation of multiple kernel learning. In Section 2.2.1 we show how multiple kernel learning can be formulated as a regularized risk minimization problem. This approach is widely known as *one-stage* multiple kernel learning in the literature. Thereafter, in Section 2.2.2, we present a different problem formulation for multiple kernel learning that is known as *two-stage* multiple kernel learning. For simplicity of notation, we assume that the set of base kernels is finite. However, it is not hard to define similar formulations for continuously-parameterized kernel sets. In methods based on one-stage MKL, kernel coefficients and a kernel-based predictor are learned simultaneously. In methods based on two-stage MKL, the kernel learning phase is separate from the predictor learning phase. This provides the flexibility of using different objective functions in each phase. However, as we will see later, this might introduce overfitting in some problems.

2.2.1 One-stage multiple kernel learning

In this section, we give the formal definition of multiple kernel learning known in the literature as one-stage multiple kernel learning. We assume that a finite set of positive semidefinite kernels is provided. Let $\mathcal{I}$ denote a finite index set, indexing the kernels or alternatively the corresponding feature maps to be combined. Let $\kappa_i : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $i \in \mathcal{I}$ denote the i^{th} kernel function. The goal is to learn a non-negative linear combination of these kernels,

$$\kappa_\theta = \sum_{i\in\mathcal{I}} \theta_i \kappa_i,$$

and use the resulting kernel to learn a predictor. Note that κ_θ is a positive semidefinite kernel since the base kernels are combined with non-negative weights. When the base kernels are linearly combined, the underlying feature maps are concatenated scaled by the square root of θ_i weights (see, e.g., Shawe-Taylor and Cristianini, 2004). This means that the set of predictors considered over the input space $\mathcal{X}$ has the form

$$\left\{ f_{u,\theta} : \mathcal{X} \to \mathbb{R} \,:\, f_{u,\theta}(x) = \sum_{i\in\mathcal{I}} \left\langle u_i, \sqrt{\theta_i}\phi_i(x) \right\rangle, \; x \in \mathcal{X}, \; u_i \in \mathcal{U}_i, \; \forall i \right\}.$$

Here, $\mathcal{U}_i$ is a Hilbert space corresponding to kernel κ_i, $\phi_i : \mathcal{X} \to \mathcal{U}_i$ is the feature-map underlying kernel κ_i, and $\langle x, y \rangle$ is the inner product over the Hilbert space that x and y belong to. The problem we consider is to solve the kernel-based optimization problem when base kernels κ_i are to be combined too:

$$\begin{aligned} \min_{u\in\mathcal{U},\theta} \quad & \text{Loss}_n(f_{u,\theta}) + \frac{\lambda}{2} \sum_{i\in\mathcal{I}} \|u_i\|_2^2, \\ \text{s.t.} \quad & \theta \geq 0, \quad \|\theta\|_p \leq C, \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} u &= (u_i)_{i\in\mathcal{I}} \in \mathcal{U} \overset{\text{def}}{=} \times_{i\in\mathcal{I}} \mathcal{U}_i, \\ \theta &\in \mathbb{R}^{\mathcal{I}}, \end{aligned}$$

$\lambda, C > 0$ are regularization parameters, and $\text{Loss}_n(f_{u,\theta})$ is the empirical loss function defined by

$$\text{Loss}_n(f_{u,\theta}) = \frac{1}{n}\sum_{t=1}^{n} \ell_t(f_{u,\theta}),$$

where $\ell_t : \mathbb{R} \to \mathbb{R}$ ($t \in \{1, \dots, n\}$) are convex loss functions. In supervised learning problems $\ell_t(y) = \ell(y_t, y)$ for some loss function $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, e.g., the squared-loss, $\ell(y_t, y) = \frac{1}{2}(y - y_t)^2$, or the hinge-loss, $\ell_t(y_t, y) = \max(1 - yy_t, 0)$, where in the former case $y_t \in \mathbb{R}$, and in the latter case $y_t \in \{-1, +1\}$. To ensure convexity of $\ell_t(\cdot)$ we assume that the loss function $\ell(\cdot, \cdot)$ is convex in its second argument.

Problem (2.4) is not jointly convex in u and θ. However, one can convert it to a convex problem by letting $w_i = \sqrt{\theta_i} u_i$. With this change the problem formulation becomes

$$\begin{aligned} \min_{w,\theta} \quad & \frac{1}{n}\sum_{t=1}^{n} \ell_t\left(\sum_{i\in\mathcal{I}} \langle w_i, \phi_i(x_t)\rangle\right) + \frac{\lambda}{2}\sum_{i\in\mathcal{I}} \frac{\|w_i\|_2^2}{\theta_i}, \\ \text{s.t.} \quad & \theta \geq 0, \quad \|\theta\|_p \leq C. \end{aligned} \tag{2.5}$$

We use the convention that

$$\frac{a}{0} = \begin{cases} 0 & \text{if} \quad a = 0, \\ \infty & \text{otherwise.} \end{cases}$$

This implies that $w_i = 0$ if $\theta_i = 0$. Problem (2.5) is a convex problem. Note that there are two tuning parameters: λ and C. Kloft et al. (2011, Theorem 1) showed that one can equivalently solve the multiple kernel learning optimization problem with a single tuning parameter:

$$\begin{aligned} \min_{w,\theta} \quad & \frac{1}{n}\sum_{t=1}^{n} \ell_t\left(\sum_{i\in\mathcal{I}} \langle w_i, \phi_i(x_t)\rangle\right) + \frac{\lambda}{2}\sum_{i\in\mathcal{I}} \frac{\|w_i\|_2^2}{\theta_i}, \\ \text{s.t.} \quad & \theta \geq 0, \quad \|\theta\|_p \leq 1. \end{aligned} \tag{2.6}$$

Problem (2.6) represents the optimization problem of *one-stage* multiple kernel learning. It is not hard to see that solving the optimization problem (2.6) with a single tuning parameter gives the same predictor as that of the optimization problem (2.5). Let (w^*, θ^*) be the optimal solution to problem (2.5) corresponding to (λ, C). In this case, $(w^*, \tilde{\theta} = \theta^*/C)$ is the optimal solution to problem (2.6) for $\tilde{\lambda} = \lambda/C$. Since the prediction function solely depends on w^*, the prediction function obtained by solving problem (2.5) for (λ, C) can be achieved by solving problem (2.6) for λ/C.

One-stage MKL can also be viewed as a group p-norm penalized minimization problem. To show this, let us define the set of predictors considered over the input space $\mathcal{X}$ as

$$\left\{ f_w : \mathcal{X} \to \mathbb{R} \, : \, f_w(x) = \sum_{i \in \mathcal{I}} \langle w_i, \phi_i(x) \rangle \, , \; x \in \mathcal{X}, \; w_i \in \mathcal{W}_i, \; \forall i \right\}$$

Here, $\mathcal{W}_i$ is a Hilbert space over the reals, $\phi_i : \mathcal{X} \to \mathcal{W}_i$ is a feature-map, and $w = (w_i)_{i \in \mathcal{I}} \in \mathcal{W} \overset{\text{def}}{=} \times_{i \in \mathcal{I}} \mathcal{W}_i$ (as an example, $\mathcal{W}_i$ may just be a finite dimensional Euclidean space). We consider the optimization problem

$$\min_{w \in \mathcal{W}} \qquad \text{Loss}_n(f_w) + \text{Pen}(f_w), \tag{2.7}$$

where $\text{Pen}(f_w)$ is a penalty term that will be specified later, and $\text{Loss}_n(f_w)$ is the empirical risk of predictor f_w. We note in passing that for the sake of simplicity, we shall sometimes abuse notation and write $\ell_n(w)$ for $\ell_n(f_w)$ and even drop the index n when the sample-size is unimportant.

We consider the special case of (2.7) when the penalty is a so-called group p-norm penalty with $1 \le p \le 2$, a case considered earlier, e.g., by Kloft et al. (2011). Thus the goal is to solve

$$\min_{w \in \mathcal{W}} \; \text{Loss}_n(w) + \frac{1}{2} \left(\sum_{i \in \mathcal{I}} \rho_i^p \|w_i\|_2^p \right)^{\frac{2}{p}} , \tag{2.8}$$

where the scaling factors $\rho_i > 0, i \in \mathcal{I}$, are assumed to be given.

The rationale of using the squared weighted p-norm is that for $1 \le p < 2$ it is expected to encourage sparsity at the group level which should allow one to handle cases when $\mathcal{I}$ is very large. The actual form, however, is also chosen for reasons of computational convenience. In fact, the reason to use the 2-norm of the weights is to allow the algorithm to work even with infinite-dimensional feature vectors (and thus weights) by resorting to the kernel trick. To see how this works, just notice that the penalty in (2.8) can also be written as

$$\left(\sum_{i \in \mathcal{I}} \rho_i^p \|w_i\|_2^p \right)^{\frac{2}{p}} = \inf \left\{ \sum_{i \in \mathcal{I}} \frac{\rho_i^2 \|w_i\|_2^2}{\theta_i} \; : \; \theta \in \Delta_{\frac{p}{2-p}} \right\} ,$$

where for $\nu \ge 1$,

$$\Delta_\nu = \left\{ \theta \in [0,1]^{|\mathcal{I}|} : \|\theta\|_\nu \le 1 \right\}$$

is the positive quadrant of the $|\mathcal{I}|$-dimensional ℓ^ν-ball (See, e.g., Micchelli and Pontil, 2005, Lemma 26). Hence, an equivalent[1] form of problem (2.8) is

$$\min_{w\in\mathcal{W},\theta\in\Delta_\nu} J(w,\theta) = \text{Loss}(w) + \frac{1}{2}\sum_{i\in\mathcal{I}} \frac{\rho_i^2 \|w_i\|_2^2}{\theta_i}, \tag{2.9}$$

where

$$\nu = \frac{p}{2-p} \in [1,\infty).$$

Let $\kappa_i : \mathcal{X}\times\mathcal{X} \to \mathbb{R}$ be the reproducing kernel underlying ϕ_i:

$$\kappa_i(x,x') = \left\langle \phi_i(x), \phi_i(x') \right\rangle, \quad x, x' \in \mathcal{X},$$

and let $\mathcal{H}_i = H_{\kappa_i}$ be the corresponding RKHS. Then, for any given fixed value of θ, the above problem becomes an instance of a standard penalized learning problem in the RKHS $\mathcal{H}_\theta$ underlying the kernel

$$\kappa_\theta = \sum_{i\in\mathcal{I}} \frac{\theta_i}{\rho_i^2}\kappa_i\,.$$

In particular, by the theorem on page 353 in Aronszajn (1950), the problem of finding $w \in \mathcal{W}$ for fixed θ is equivalent to

$$\min_{f\in\mathcal{H}_\theta} \text{Loss}(f) + \frac{1}{2}\|f\|_{\mathcal{H}_\theta}^2,$$

and thus (2.8) is equivalent to

$$\min_{f\in\mathcal{H}_\theta,\theta\in\Delta_\nu} \text{Loss}(f) + \frac{1}{2}\|f\|_{\mathcal{H}_\theta}^2\,.$$

Thus, we see that the method can be thought of as finding the weights of a kernel κ_θ and a predictor minimizing the $\mathcal{H}_\theta$-norm penalized empirical risk. This shows that the group p-norm optimization problem presented above is an instance of multiple kernel learning. In the next section, we present a two-stage problem formulation for multiple kernel learning.

2.2.2 Two-stage multiple kernel learning

In one-stage multiple kernel learning the kernel function and the predictor are learned simultaneously in a single optimization problem. The two-stage problem formulation, on the other hand, consists of two steps. In the first step the kernel

[1]Here and in what follows by equivalence we mean that the set of optimums in terms of w (the primary optimization variable) is the same in the two problems.

function is learned, usually by solving an optimization problem. Then, in the second step, the resulting kernel from the first step is used with a kernel-based method (e.g. SVM or ridge regression) to learn a predictor. The key component to the methods in this family is the surrogate objective function used to learn the kernel function. The objective function in the first step should be a measure of the quality or "goodness"of a given kernel function.

In this section, we describe a well-known objective function for the two-stage problem formulation: *centered alignment* between the learned kernel and the "ideal" kernel. The ideal kernel underlying the common distribution of the data is

$$\kappa^*(x, x') = \mathbb{E}\left[\, YY' \,|\, X = x, X' = x' \,\right].$$

In two-stage methods the goal is to learn a kernel combination that is maximally "aligned" to the ideal kernel. The objective function that we consider here is the centered alignment metric proposed by Cortes et al. (2010), which originated from the definition of alignment proposed by Cristianini et al. (2002).[2] Centered alignment as a metric to measure the similarity of two kernel functions, κ, and $\tilde{\kappa}$ is defined by:

$$A_c(\kappa, \tilde{\kappa}) \stackrel{\text{def}}{=} \frac{\langle \kappa_c, \tilde{\kappa}_c \rangle}{\|\kappa_c\| \|\tilde{\kappa}_c\|},$$

where κ_c is the kernel underlying κ centered in the feature space (similarly for $\tilde{\kappa}_c$), $\langle \kappa, \tilde{\kappa} \rangle = \mathbb{E}\left[\kappa(X, X')\tilde{\kappa}(X, X')\right]$ and $\|\kappa\|^2 = \langle \kappa, \kappa \rangle$. A kernel κ centered in the feature space, by definition, is the unique kernel κ_c, such that for any x, x',

$$\kappa_c(x, x') = \left\langle \phi(x) - \mathbb{E}\left[\phi(X)\right], \phi(x') - \mathbb{E}\left[\phi(X)\right] \right\rangle,$$

where ϕ is a feature map underlying κ. By considering centered kernels κ_c, $\tilde{\kappa}_c$ in the alignment metric, one implicitly matches the mean responses $\mathbb{E}[\kappa(X, X')]$, and $\mathbb{E}[\tilde{\kappa}(X, X')]$ before considering the alignment between the kernels (thus, centering depends on the distribution of x). An alternative way of stating this is that centering cancels mismatches of the mean responses between the two kernels. When one of the kernels is the ideal kernel, centered alignment effectively standardizes the alignment by cancelling the effect of imbalanced class distributions. For further discussion of the virtues of centered alignment, see Cortes et al. (2010).

Since the common distribution underlying the data is unknown, given a dataset $\{(x_i, y_i)\}_{i=1}^n$ one resorts to empirical approximations to alignment and centering,

[2] Note that the word metric is used in its everyday sense and not in its mathematical sense.

resulting in the empirical alignment metric,

$$\hat{A}_c(\mathbf{K}, \tilde{\mathbf{K}}) = \frac{\left\langle \mathbf{K}_c, \tilde{\mathbf{K}}_c \right\rangle_F}{\|\mathbf{K}_c\|_F \|\tilde{\mathbf{K}}_c\|_F},$$

where,

$$\begin{aligned} \mathbf{K} &= (\kappa(x_i, x_j))_{1 \le i,j \le n}, \quad \text{and} \\ \tilde{\mathbf{K}} &= (\tilde{\kappa}(x_i, x_j))_{1 \le i,j \le n} \end{aligned}$$

are the kernel matrices underlying κ and $\tilde{\kappa}$, and for a kernel matrix, $\mathbf{K}$,

$$\mathbf{K}_c = \mathbf{C}_n \mathbf{K} \mathbf{C}_n,$$

where $\mathbf{C}_n$ is the so-called centering matrix defined by

$$\mathbf{C}_n = \mathbf{I}_{n \times n} - \frac{1}{n} \mathbf{1}\mathbf{1}^\top,$$

$\mathbf{I}_{n \times n}$ being the $n \times n$ identity matrix and $\mathbf{1} = (1, \ldots, 1)^\top \in \mathbb{R}^n$. The empirical counterpart of maximizing $A_c(\kappa, \kappa^*)$ is to maximize $\hat{A}_c(\mathbf{K}, \hat{\mathbf{K}}^*)$, where $\hat{\mathbf{K}}^* \overset{\text{def}}{=} \mathbf{y}\mathbf{y}^\top$, where $\mathbf{y} = (y_1, \ldots, y_n)^\top$ collects the responses into an n-dimensional vector. Here, $\mathbf{K}$ is the kernel matrix derived from a kernel κ. To make this connection clear, we will write $\mathbf{K} = \mathbf{K}(\kappa)$ and $\mathbf{K}_i = \mathbf{K}(\kappa_i)$. Define $f : \mathcal{K} \to \mathbb{R}$ by $f(\kappa) = \hat{A}_c(\mathbf{K}(\kappa), \hat{\mathbf{K}}^*)$, where $\mathcal{K}$ is the space of positive semidefinite kernels. When $\mathcal{K}$ is restricted to the non-negative linear combination of a given set of base kernels, indexed by $\mathcal{I}$, the problem of centered alignment maximization is given by

$$\max_{\theta \ge 0} \quad \frac{\left\langle \sum_{i \in \mathcal{I}} \theta_i \mathbf{K}_i, \hat{\mathbf{K}}^* \right\rangle_F}{\| \sum_{i \in \mathcal{I}} \theta_i \mathbf{K}_i \|_F}. \tag{2.10}$$

Problem (2.10) is not convex since centered alignment is not a convex function. In the next section we show that there exists a convex optimization problem that is equivalent to alignment maximization.

Alignment maximization by using a convex objective function

Instead of using alignment as the objective function of a two-stage problem one can resort to minimizing the L_2 distance between the kernel combination and the ideal kernel. Note that the objective function in (2.10) is scale invariant, i.e. if θ^* is a solution, then any $\tilde{\theta} = \nu\theta^*, \nu > 0$ is also a solution. Therefore, one can solve (2.10) by just maximizing the numerator (or minimizing the negated numerator), while a constraint is imposed on its denominator. Let $b_i = \langle \mathbf{K}_i, \hat{\mathbf{K}}^* \rangle_F$, and $\mathbf{M}_{ij} = \langle \mathbf{K}_i, \mathbf{K}_j \rangle_F$, $i, j \in \mathcal{I}$. While problem (2.10) is not convex, one can equivalently solve the following convex problem:

$$\begin{aligned} \min_{\theta \geq 0} \quad & -b^\top \theta \\ \text{s.t.} \quad & \theta^\top \mathbf{M} \theta \leq 1. \end{aligned} \tag{2.11}$$

We will show that the solution to problem

$$\min_{\mu \geq 0} \quad f_C(\mu) = -b^\top \mu + C\mu^\top \mathbf{M}\mu, \tag{2.12}$$

for every $C > 0$ is a solution to problem (2.11) once scaled properly.

Proof. Let μ^* be the optimal solution to problem (2.12). We show, by contradiction, that $\theta^* = \frac{1}{\sqrt{\Delta}}\mu^*$ is the optimal solution to problem (2.11), where $\Delta = {\mu^*}^\top \mathbf{M}\mu^*$. Suppose θ^* is not the optimal solution to problem (2.11), i.e. there exists $u \geq 0$ such that $u^\top \mathbf{M} u = 1$, and $-b^\top u < -b^\top \theta^*$. Let $w = \sqrt{\Delta} u$. We have $w^\top \mathbf{M} w = \mu^* \mathbf{M} \mu^*$, and $-b^\top w < -b^\top \mu^*$. This implies that μ^* is not the optimal solution to problem (2.12), which contradicts the initial assumption. Therefore θ^* is the optimal solution to problem (2.11). □

Effect of parameter C on the rate of convergence of alignment

We explore the effect of parameter C in problem (2.12) on the rate of convergence of alignment for gradient-type algorithms since we are interested in such algorithms in this thesis. We will show that different values of $C > 0$ result in the same trajectory of alignment values. Assume that a projected gradient descent algorithm is applied to solve problem (2.12) for two different values: $C, \tilde{C} > 0$. Let

$$\{\mu_0, \mu_1, \ldots, \mu_t, \ldots\} \quad \text{and} \quad \{\tilde{\mu}_0 = \frac{C}{\tilde{C}}\mu_0, \tilde{\mu}_1, \ldots, \tilde{\mu}_t, \ldots\}$$

be the trajectories obtained by the projected gradient descent for C and $\tilde{C}$ respectively. Note that we assume that $\tilde{\mu}_0 = \frac{C}{\tilde{C}}\mu_0$. This is not a very restrictive assumption. In fact this assumption holds if the standard initialization of $\mu_0 = 0$ is used. For simplicity we aim to optimize $\frac{1}{2C} f_C(\mu)$. The update rule yields

$$\begin{aligned} \mu_{t+1} &= \Pi\left(\mu_t - \frac{\eta_t}{2C}\nabla f_C(\mu_t)\right) = \Pi\left(\mathbf{Z}_t \mu_t + \frac{\eta_t}{2C} b\right), \quad \text{and} \\ \tilde{\mu}_{t+1} &= \Pi\left(\mathbf{Z}_t \tilde{\mu}_t + \frac{\eta_t}{2\tilde{C}} b\right), \end{aligned}$$

where $\nabla f_C(\mu_t) = -b + 2C\mathbf{M}\mu_t$, $\mathbf{Z}_t = \mathbf{I} - \eta_t \mathbf{M}$, $\mathbf{I}$ is the identity matrix of suitable size, η_t is the learning rate at iteration t, and Π is the orthogonal projection operator onto the region $\mu \geq 0$. For a vector v, the projection is obtained by $\Pi(v) = \mathbf{S}_v v$, where

$$[\mathbf{S}_v]_{ij} = \begin{cases} 1 & i = j, \quad \text{and} \quad v_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

Recall that $\tilde{\mu}_0 = \frac{C}{\tilde{C}}\mu_0$. Assume $\tilde{\mu}_t = \frac{C}{\tilde{C}}\mu_t$. We show that $\tilde{\mu}_{t+1} = \frac{C}{\tilde{C}}\mu_{t+1}$.

$$\begin{aligned} \tilde{\mu}_{t+1} &= \Pi\left(\mathbf{Z}_t\tilde{\mu}_t + \frac{\eta_t}{2\tilde{C}}b\right) \\ &= \Pi\left(\frac{C}{\tilde{C}}\mathbf{Z}_t\mu_t + \frac{\eta_t}{2\tilde{C}}b\right) \\ &= \Pi\left(\frac{C}{\tilde{C}}\left(\mathbf{Z}_t\mu_t + \frac{\eta_t}{2C}b\right)\right) \\ &= \frac{C}{\tilde{C}}\Pi\left(\mathbf{Z}_t\mu_t + \frac{\eta_t}{2C}b\right) \\ &= \frac{C}{\tilde{C}}\mu_{t+1}. \end{aligned}$$

Note that by the definition of Π, we have $\Pi(\frac{C}{\tilde{C}}v) = \frac{C}{\tilde{C}}\Pi(v)$. We showed that the trajectory obtained by the projected gradient method for $\tilde{C}$ is the same as the trajectory obtained for C, scaled by $\frac{C}{\tilde{C}}$. Since the alignment objective function is invariant to scaling, we conclude that the two trajectories yield the same alignment values.

2.3 Related Work

In this section, we review some of the previous approaches to MKL that are most related to the topic of this thesis. In particular, we focus on algorithms that were designed to handle large sets of kernels. We divide the relevant work in MKL into two main categories of one-stage and two-stage MKL algorithms that we will review in Sections 2.3.1 and 2.3.2 respectively.

2.3.1 One-stage MKL methods

A large body of the MKL literature has been dedicated to address the problem of one-stage MKL. We divide these methods into three main categories:

- Methods for learning to combine moderate-size kernel sets
- Methods for learning to combine large kernel sets

- Methods for learning to combine infinitely-many kernels

In the next sections we review some of the well-known algorithms in each category.

Methods for learning to combine moderate-size kernel sets

In one of the pioneering papers in MKL, Lanckriet et al. (2004) present a semidefinite programming (SDP) problem formulation for learning a positive semidefinite kernel matrix over training and test data in a transductive setting, where it is assumed that test data (for which we are to determine the labels) are available. Then, they proposed a quadratically constrained quadratic programming (QCQP) optimization problem when the goal is to learn a non-negative linear combination of a finite set of base kernels. For soft-margin SVM the QCQP problem formulation is

$$\begin{aligned}
\max_{\alpha \in \mathbb{R}^n, \tau} \quad & 2\alpha^\top \mathbf{1} - \varepsilon \alpha^\top \alpha - \tau \\
\text{s.t.} \quad & \tau \geq \frac{1}{z_i} (\alpha \circ \mathbf{y})^\top \mathbf{K}_i (\alpha \circ \mathbf{y}), \quad i = 1, \dots, r \\
& \alpha^\top \mathbf{y} = 0, \\
& 0 \leq \alpha \leq C,
\end{aligned}$$

where $\varepsilon > 0$, $r = |\mathcal{I}|$, $z_i = \operatorname{tr}(\mathbf{K}_i)$ and $\circ$ denotes element-wise product.

The solvers that use interior-point methods yield a worst-case complexity of (roughly) $O((|\mathcal{I}| + n)^2 n^{2.5})$ for SDP and $O(|\mathcal{I}| n^3)$ for QCQP problems for MKL (Lanckriet et al., 2004). These algorithms become intractable when the number of training examples or the size of the kernel set increase. In the experiments, Lanckriet et al. (2004) considered combining only a few kernels over small datasets. Hence, most of the subsequent work in MKL has focused on designing faster and more scalable algorithms. For instance, Sonnenburg et al. (2006) proposed an iterative algorithm for MKL in which, in each iteration, kernel weights are updated through a semi-infinite linear program of the following structure:

$$\begin{aligned}
\max_{\tau \in \mathbb{R}, \theta} \quad & \tau \\
\text{s.t.} \quad & \theta \geq 0, \quad \|\theta\|_1 = 1, \\
& \sum_{i \in \mathcal{I}} \theta_i S_i(\alpha) \geq \tau, \quad \text{for all } \alpha \in \mathcal{A},
\end{aligned}$$

where $S_i(\alpha)$ and $\mathcal{A}$ depend on the loss function used. For instance, for the hinge loss

$$S_i(\alpha) = (\alpha \circ \mathbf{y})^\top \mathbf{K}_i (\alpha \circ \mathbf{y}) - \alpha^\top \mathbf{1},$$

and

$$\mathcal{A} = \left\{\alpha : \alpha^\top \mathbf{y} = 0,\ 0 \leq \alpha \leq C\right\}.$$

Once the kernel weights are obtained, a standard SVM algorithm is used to update the predictor weights using the linear combination of kernel matrices. This procedure is repeated until convergence is achieved. Rakotomamonjy et al. (2008) proposed a similar iterative algorithm, SimpleMKL, in which gradient descent replaces linear programming to update kernel weights. Chapelle and Rakotomamonjy (2008) proposed HessianMKL, a second order optimization approach to further improve the computational efficiency of SimpleMKL. The above algorithms generalize the problem formulation to convex loss functions, hence they are applicable to a wider range of problems including regression and one-class classification problems. Another common theme between these methods is that they focus on learning a sparse combination of kernels through penalizing 1-norm of the kernel weights.

Orabona et al. (2010) proposed a two-stage stochastic gradient descent algorithm to solve the general p-norm MKL problem. Their algorithm is an extension of the Pegasos algorithm, one of the fastest solvers for linear SVM (Shalev-Shwartz et al., 2007), to MKL. In the first stage, a fast online algorithm is used to determine the region of space where the optimal solution lies. This solution is passed to the second stage in which a stochastic proximal projected subgradient descent algorithm due to Do et al. (2009) is performed to obtain the final solution. In each iteration, their algorithm uses a random sample of training data to calculate the subgradient of the objective function. Hence, this algorithm, similar to Pegasos, is scalable to large datasets. The computational complexity of each iteration, however, scales linearly in the number of base kernels, which makes the algorithm unattractive for very large kernel sets.

One of the fastest MKL algorithms was proposed by Kloft et al. (2011). It addresses MKL problem with a general convex loss function and p-norm regularization on kernel coefficients for $p > 1$. Like previous approaches they proposed a two-step iterative algorithm. In each iteration, the kernel coefficients are updated by a closed-form update rule, and then a standard kernel method is used to learn a predictor on top of the fixed kernel coefficients. They show that given $\|w_i\|_2^2$, the kernel coefficients in each iteration can be updated by[3]

$$\theta_i = \frac{\|w_i\|_2^{\frac{2}{p+1}}}{\left(\sum_{j\in\mathcal{I}} \|w_j\|_2^{\frac{2p}{p+1}}\right)^{\frac{1}{p}}}, \quad \forall i \in \mathcal{I}.$$

They show experimentally that their method is faster than previous MKL algorithms. For instance, they showed that their algorithm achieves a speed-up of one and two orders of magnitude over HessianMKL and SimpleMKL respectively (Kloft et al., 2011).

The tradeoff between sparsity and accuracy has been studied by Tomioka and Suzuki (2010) in the context of MKL by using a regularization term similar to that of the elastic net (Zou and Hastie, 2005). More recently, Orabona and Jie (2011) proposed a fast MKL method based on a similar problem formulation. Their algorithm, UFO–MKL, uses a regularizer that consists of two terms. The first term is a squared $(2, 2\log r/(2\log r - 1))$ group norm where r is the number of kernels. The second term, however, is a $(2,1)$ group norm to induce sparsity. A parameter controls the tradeoff between these terms. Orabona and Jie (2011) showed that the rate of convergence of UFO–MKL depends logarithmically on the number of kernels. Each iteration of the algorithm, however, scales linearly in the number of kernels as the algorithm updates all coordinates at each iteration. They proposed a stochastic gradient descent algorithm to solve an optimization problem with the above strongly convex regularization term.

Methods for learning to combine large kernel sets

Many of the above methods, though efficient in handling large datasets, fail to scale for large kernel sets. These algorithms update the entire kernel weight vector at every iteration (see, e.g., Sonnenburg et al., 2006; Rakotomamonjy et al., 2008; Kloft et al., 2011; Orabona and Jie, 2011). Hence, their computational complexity depends linearly on the number of kernels. The highest number of base kernels considered in an experiment was reported in Kloft et al. (2011) in which they perform MKL to combine 10^4 kernels. These methods do not scale well when one considers very large kernel sets. As we will discuss later in Chapter 6, having an exponential number of base kernels is not an unrealistic scenario. There are, however, a few algorithms

[3]The analytical optimization method proposed in Kloft et al. (2011) was independently proposed by Xu et al. (2010) too.

designed specifically to handle very large kernel sets by exploiting the structure of kernel space.

Bach (2008) introduces *hierarchical* MKL in which the base kernels are arranged as nodes in a directed acyclic graph. The primary focus of this method is on kernels that can be expressed as "product of sums". One prominent example of such kernels is polynomial kernels expressed as

$$\kappa(x,z) = \prod_{i=1}^{r} (1 + x_i z_i)^D .$$

In this example the nodes are tuples of length r, and each node $(j_1, \ldots, j_r)$ is connected to $(j_1+1, j_2, \ldots, j_r)$, $\ldots$, $(j_1, \ldots, j_{r-1}, j_r+1)$. The objective of hierarchical MKL is to select and add a product of kernels to the combination only after all of the subproducts are selected. Let $G = (V, E)$ be the graph of kernels in which V is the set of nodes (kernels) and $E \subset V \times V$ be the set of edges. For a node $v \in V$, let $A(v)$ denote the ancestors of v. By convention $v \in A(v)$. For a subset of nodes $I \subset V$, the *hull* of I is defined as the union of all ancestors of $v \in I$: $\mathrm{hull}(I) = \bigcup_{v \in I} A(v)$. Note that the hull of a set I is also the complement of the set of nodes that are descendants of I^c, where I^c denotes the complement of I. The kernel selection property mentioned above implies that if a node is selected then its hull must be in the set of selected nodes too, since to include a node the entire set of ancestors must be selected. To estimate $\mathrm{hull}(I)$ one must look for vertices $v \in V$ such that $D(v) \subset I^c$, where $D(v)$ is the set of descendants of v. Bach (2008) thus considered the following 1-norm penalty term

$$\sum_{v \in V} d_v \|w_{D(v)}\| = \sum_{v \in V} d_v \left(\sum_{u \in D(v)} \|w_u\|^2 \right)^{1/2} , \tag{2.13}$$

where $(d_v)_{v \in V}$ are some positive user-defined weights. Penalizing by (2.13) will impose that some of the vectors $w_{D(v)}$ are zero. Hence, the optimization problem considered by Bach (2008) is

$$\min_{w} \quad \frac{1}{n} \sum_{t=1}^{n} \ell_t \left(\sum_{v \in V} \langle w_v, \phi_v(x_t) \rangle \right) + \frac{\lambda}{2} \left(\sum_{v \in V} d_v \|w_{D(v)}\| \right)^2 . \tag{2.14}$$

The reformulation of (2.14) is done as follows: Let $\eta = (\eta_v)_{v \in V}$ be a vector of $|V|$ elements such that $\eta \geq 0$ and $\sum_{v \in V} d_v^2 \eta_v \leq 1$. Let

$$\zeta_v(\eta) = \left(\sum_{v' \in A(v)} \eta_{v'}^{-1} \right)^{-1} .$$

Define the kernel combination by

$$\kappa_\eta(x,z) = \sum_{v \in V} \zeta_v(\eta)\kappa_v(x,z)\,.$$

Bach (2008) shows that with some change of variables problem (2.14) is equivalent to the MKL problem

$$\begin{aligned}
\min_{w,\eta} \quad & \frac{1}{n}\sum_{t=1}^{n} \ell_t\left(\sum_{v \in V}\langle w_v, \phi_v(x_t)\rangle\right) + \frac{\lambda}{2}\sum_{v\in V}\frac{\|w_v\|^2}{\zeta_v(\eta)} \\
\text{s.t.} \quad & \eta \geq 0, \quad \sum_{v \in V} d_v^2 \eta_v \leq 1\,.
\end{aligned} \tag{2.15}$$

Bach (2008) then proposes an efficient algorithm to solve (2.15). The computational complexity of his algorithm is polynomial in the number of *selected* kernels, and is independent of the total number of kernels. This provides a tractable method to learn polynomial kernels when the kernel set has exponentially many kernels. The experiments conducted in Bach (2008) show that this algorithm performs well in accuracy and computational efficiency, especially when the number of kernels is very large.

A quite different approach was taken by Cortes et al. (2009b) for the problem of learning polynomial kernels by simplifying the form of the resulting kernels. They proposed a restricted form of polynomial kernels that results in a dramatic reduction in the number of parameters to be learned. For a given set of base kernels, $\{\kappa_1, \ldots, \kappa_r\}$ the resulting kernel is of the form,

$$\begin{aligned}
\kappa_\theta(x,z) &= \sum_{i_1,\ldots,i_D=1}^{r} \theta_{i_1}\ldots\theta_{i_D}\kappa_{i_1}(x,z)\ldots\kappa_{i_D}(x,z) \\
&= \left(\sum_{i=1}^{r}\theta_i\kappa_i(x,z)\right)^D,
\end{aligned} \tag{2.16}$$

where $\theta \in \mathbb{R}^r$ is a non-negative parameter vector, and $D > 0$ is a user-defined integer that determines the degree of the resulting polynomial kernel. They proposed a projected gradient descent algorithm to solve the optimization problem for kernel ridge regression:

$$\begin{aligned}
\min_{\theta} \quad & \mathbf{y}^\top(\mathbf{K}_\theta + \lambda I)^{-1}\mathbf{y}, \\
\text{s.t.} \quad & \theta \geq 0, \quad \|\theta - \theta_0\|_2 \leq \Lambda\,,
\end{aligned} \tag{2.17}$$

where θ_0 (offset) and $\Lambda > 0$ are user-defined parameters. Note that problem (2.17) with the parameterization given in (2.16) is not convex .

Methods for learning to combine infinitely-many kernels

All of the above algorithms are designed to combine a finite set of kernels. If continuously parameterized kernels, such as Gaussian kernels, are of interest, the standard approach is to select a predetermined number of base kernels from a bounded interval by discretizing the interval (see, e.g., Rakotomamonjy et al. (2008); Kloft et al. (2011)). There are inherent flaws with the discretization of such kernel spaces. One problem is that many kernels will be left out from the kernel set, and if some of these kernels are required to obtain high accuracy the resulting predictor might perform poorly. We will study this problem in Chapter 3. Another flaw of this approach is that while it may be computationally feasible for one-dimensional parameterizations, discretizing a space of multi-dimensional parameters results in a large number of base kernels that is exponential in the number of parameters.

A computationally feasible approach for continuously parameterized kernels was first introduced by Argyriou et al. (2005). They proposed a coordinate descent approach that greedily selects a kernel parameter at each iteration. They empirically demonstrated that combining kernels chosen from a continuous parameter space results in better performance than discretization. We will study this algorithm as greedy coordinate descent for one-stage MKL in Chapter 5. One of the challenges of this method is that it solves a non-convex subproblem to find the best kernel (parameter) at each iteration. Argyriou et al. (2006) proposed a DC-programming algorithm for MKL to address the local minima issue in learning Gaussian kernels. The new approach is, to some extent, faster in one-dimensional kernel parameter search, however, it does not scale well when the number of kernel parameters increases. Gehler and Nowozin (2008) proposed a similar algorithm for learning to combine kernels from continuously parameterized sets. Their proposed method was very similar to that of Argyriou et al. (2005), except that it performs a totally-corrective scheme at the end of each iteration, i.e., all kernel weights are recomputed once a new kernel is added to the combination. They did not give any convergence guarantees for their method.

2.3.2 Two-stage MKL methods

The approach taken by the one-stage MKL methods is to define a joint optimization problem to learn a kernel combination and predictor simultaneously. A second approach to MKL, which is called *two-stage* MKL, separates kernel learning from predictor learning. In the first stage, the kernel function is learned through the use of a surrogate objective function, such as alignment (Cristianini et al., 2002) or a modified version of it, i.e., centered alignment (Cortes et al., 2010), that measures the *similarity* of two kernel functions over the training data. For the purpose of MKL, alignment is usually measured between the kernel combination and the so-called *target label* kernel or the *ideal* kernel over the training data, $\mathbf{K}^* = \mathbf{y}\mathbf{y}^\top$, where $\mathbf{y} = (y_1, \ldots, y_n)^\top$ is the vector of training labels. The learned kernel combination is then used in the second stage, using standard kernel methods, such as SVM or kernel ridge regression to learn a predictor. While this approach is modular and simple, unless we use a mechanism to control the complexity of kernel, there is a chance of overfitting to the objective function of the first stage. We investigate this phenomenon in Chapter 6.

Learning a kernel function by optimizing the alignment measure has been the main theme of two-stage MKL. For instance, Kandola et al. (2002) proposed a method for optimizing alignment over a combination of kernels in both transductive and inductive settings based on the incomplete Cholesky matrix factorization of the kernel matrix. Igel et al. (2007) proposed a gradient-based optimization of alignment. He et al. (2008) proposed to minimize the Euclidean distance between a non-negative linear combination of base kernels and the ideal kernel. They proposed a quadratic programming (QP) optimization problem to learn the kernel weights. While they did not mention any connection between their problem formulation and alignment maximization, as we demonstrated in Section 2.2 the two problems are indeed equivalent.

Alignment has been used widely as a measure of similarity between kernels in two-stage MKL methods. However, it is not the only measure. For example, Nguyen and Ho (2008) introduced a new measure that uses data distributions in the feature space. The new measure, called *feature space-based kernel matrix evaluation measure* (FSM) is defined as the ratio of the total within-class standard deviation in the direction between the class centers to the distance between the class centers. For a two-class classification problem let n_+ and n_- denote the number of training examples in the two classes. Let ϕ_+ and ϕ_- denote the center of the positive and negative classes respectively:

$$\begin{aligned}\phi_+ &= \frac{1}{n_+}\sum_{i:y_i=1}\phi(x_i),\\ \phi_- &= \frac{1}{n_-}\sum_{i:y_i=-1}\phi(x_i).\end{aligned}$$

Then the FSM measure is defined by

$$FSM(\mathbf{K},\mathbf{y}) \overset{\text{def}}{=} \frac{std}{\|\phi_- - \phi_+\|},$$

where

$$std = \sqrt{\frac{1}{n_+ - 1}\sum_{i:y_i=1}\langle\phi(x_i)-\phi_+, u\rangle^2} + \sqrt{\frac{1}{n_- - 1}\sum_{i:y_i=-1}\langle\phi(x_i)-\phi_-, u\rangle^2},$$

where

$$u = \frac{\phi_- - \phi_+}{\|\phi_- - \phi_+\|}$$

is the unit vector in the direction of between-class centers. Nguyen and Ho (2008) compared alignment and FSM in a series of experiments and concluded that while alignment has some fundamental limitations, FSM reflects better the error rates of SVMs. Later, Tanabe et al. (2008) proposed to optimize kernel coefficients on a simplex using the FSM similarity measure. Among other works, Ying et al. (2009) proposed to learn kernel coefficients by optimizing an information-theoretic measure based on KL divergence between the combined kernel matrix and the ideal kernel. They proposed to apply gradient descent to solve the kernel coefficients over a simplex with the following optimization problem:

$$\min_{\theta\in\Delta_1} \quad \operatorname{tr}\left(\mathbf{K}^*\left(\sum_{i\in\mathcal{I}}\theta_i\mathbf{K}_i+\varepsilon\mathbf{I}\right)^{-1}\right)+\log\left|\sum_{i\in\mathcal{I}}\theta_i\mathbf{K}_i+\varepsilon\mathbf{I}\right|,$$

where $\varepsilon > 0$ is a small constant.

2.4 Summary

In this chapter we reviewed kernel methods and two popular variants of the MKL problem formulation, i.e. one-stage and two-stage MKL. In one-stage MKL, the kernel function and predictor are learned simultaneously. Two-stage MKL, on the other hand, consists of two separate stages for learning the kernel function and learning the predictor. We showed that while alignment maximization is not a convex optimization problem it can equivalently be solved by means of an equivalent convex problem. We also reviewed some of the well-known methods for one-stage and two-stage MKL. Most of these methods do not scale well when the number of base kernels increases. In Chapters 4 and 5 we will present algorithmic approaches to solve one-stage and two-stage MKL problems when one considers combining exponentially or infinitely many kernels.

Chapter 3

When Finite Multiple Kernel Learning Fails

It is not difficult to come up with examples in which current state-of-the-art MKL algorithms that are based on combining a finite set of kernels fail. One simple scenario is when each base kernel is associated with one of the input variables. In this case, the resulting predictor of these MKL algorithms is simply a linear predictor. If the true function is sufficiently non-linear, one will not be able to achieve good performance using such a kernel. One may argue that the solution is to consider non-linear features and to provide kernels associated with higher-order monomials to such MKL algorithms. While this may solve the issue in the above example, the resulting number of non-linear features, and consequently the size of kernel set, is exponential in the degree of monomials. The complexity of finite MKL algorithms, as we will demonstrate in Chapter 6, is linear in the number of base kernels, which renders these algorithms inefficient at combining exponentially many kernels. This shows the importance of choosing the right set of kernels for an MKL algorithm.

Another shortcoming of finite MKL algorithms occurs when a practitioner discretizes a continuous kernel parameter space in order to make a finite set of kernels. One prominent example is the family of Gaussian kernels parameterized by a bandwidth. If the chosen discretization is not fine enough, it is possible to miss some of the important kernels. One easy fix to this problem is to use a finer level of discretization. However, when one considers multi-dimensional kernel parameters, this results in an exponential growth in the number of kernels as a function of the number of parameters. In this chapter we will use a simple example to show the importance of having the key kernels in the kernel set.

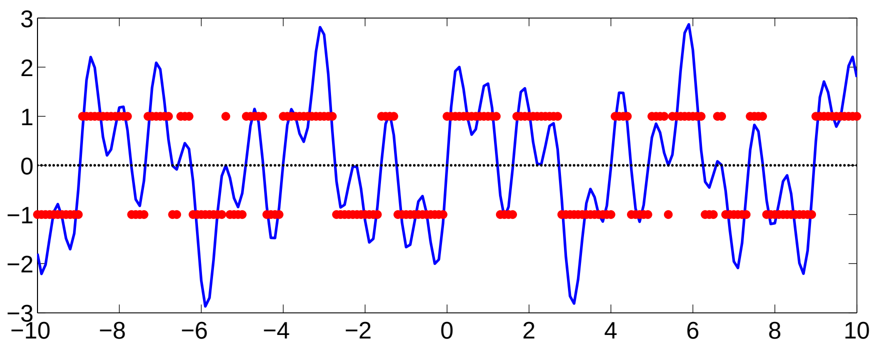

Figure 3.1: The function $f(x) = \sin(\sqrt{2}x) + \sin(\sqrt{12}x) + \sin(\sqrt{60}x)$ used for generating synthetic data, along with $\text{sign}(f)$.

3.1 Importance of Combining the Right Set of Kernels

In this section we show that it is important to combine the right dictionary of kernels to achieve good predictive accuracy. The purpose of this experiment is mainly to provide empirical proof for the following hypotheses:

- (H1) The combination of multiple kernels can lead to improved performance as compared to what can be achieved with a single kernel.
- (H2) In some cases, providing a finite set of base kernels to an MKL algorithm may fail to achieve good performance. However, methods that select kernels from continuously parameterized families are able to find the "key" kernels and their combination.

To illustrate (H1) and (H2) we have designed the following one-dimensional problem: the inputs are generated from the uniform distribution over the interval $[-10, 10]$. The label of each data point is determined by the function $y(x) = \text{sign}(f(x))$, where

$$f(x) = \sin(\sqrt{2}x) + \sin(\sqrt{12}x) + \sin(\sqrt{60}x).$$

Figure 3.1 shows functions f (blue curve) and y (red dots). Training and validation sets include 500 data points each, while the test set includes 1000 instances. In this experiment, all classifiers were trained using the soft margin SVM method, where the regularization coefficient of SVM was tuned, using the validation set, from $10^{\{-5,-4.5,\ldots,4.5,5\}}$. For this experiment we use Dirichlet kernels of degree one, parameterized with a frequency parameter σ:

$$\kappa_\sigma(x, x') = 1 + 2\cos(\sigma\|x - x'\|).$$

In order to investigate (H1), we searched through the kernel frequencies in the range $[0, 20]$. We measure misclassification error of each predictor associated with a kernel frequency. We plot error values as a function of the kernel frequency for 1000 frequencies in the above range. Figure 3.2 shows the results. We noticed that the kernel frequencies used to generate data, i.e. $\{\sqrt{2}, \sqrt{12}, \sqrt{60}\}$, marked with dashed lines in the figure, give the lowest errors. Note that each of these three kernels, if used

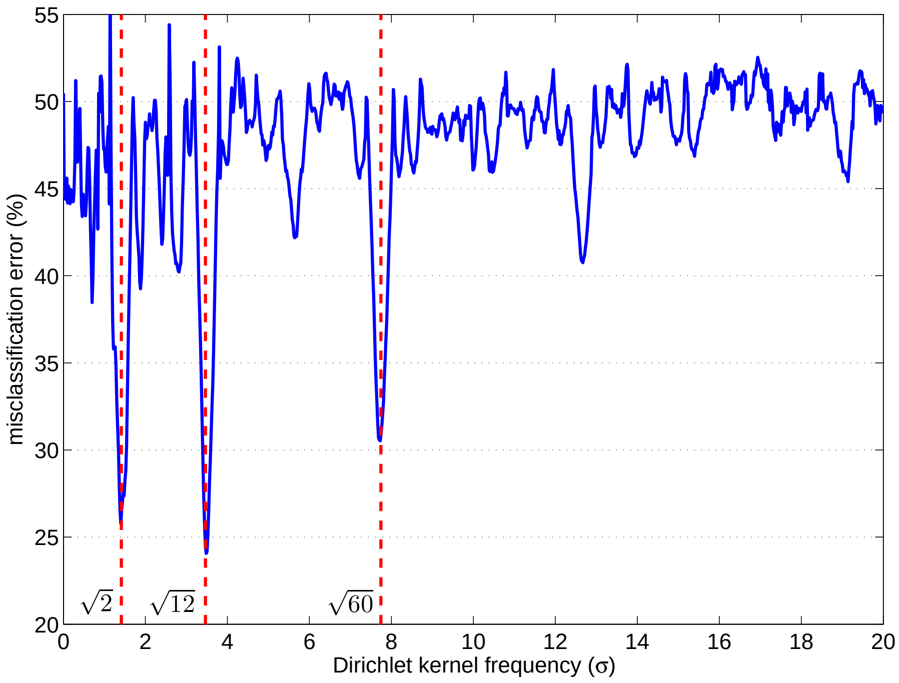

Figure 3.2: Misclassification error as a function of the kernel frequency. Dashed lines specify the three frequencies, viz. $\{\sqrt{2}, \sqrt{12}, \sqrt{60}\}$, used to generate data.

singly, cannot achieve an error rate less than 24%. Next, we measure error for pair of kernels. Classifiers trained with a pair of frequencies, i.e. $\{\sqrt{2}, \sqrt{12}\}$, $\{\sqrt{2}, \sqrt{60}\}$, and $\{\sqrt{12}, \sqrt{60}\}$ achieved error rates of 16.4%, 20.0%, and 21.3%, respectively (the kernels were combined using uniform weights). Finally, we combined all three of these kernels with uniform weights. The resulting classifier achieved an error rate of 2.3%, which is a dramatic improvement over the previous kernels alone or in combination.

Let us now turn to (H2). To verify this hypothesis we compare finite MKL algorithms against two methods that are able to combine kernels from a continuous parameter set. Finite MKL methods include AMKL (Cortes et al., 2010), LPMKL

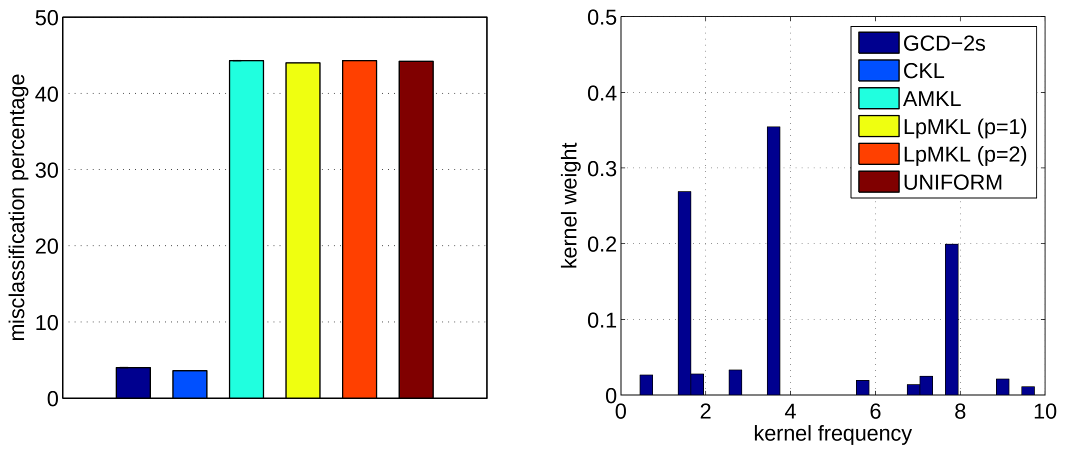

Figure 3.3: Misclassification percentages obtained by each algorithm (left), and the kernel frequencies found by the GCD–2s method (right).

(with $p = 1$, and $p = 2$) (Kloft et al., 2011), and Uniform, which is uniform combination of base kernels. The base kernels for these methods are generated by discretization of the parameter space. We choose 10 Dirichlet kernels with $\sigma \in \{0, 1, \ldots, 9\}$, covering the range of frequencies defining f. Methods that select and combine kernels from a continuous parameter set include CKL (Argyriou et al., 2005), and a new method, denoted by GCD–2s, that will be explained later in Chapter 5. As shown in Figure 3.3, GCD–2s and CKL achieved a misclassification error close to what was seen when the three best frequencies were used, showing that they are indeed capable of finding effective kernel combinations. Furthermore, the right plot in Figure 3.3 shows that the frequencies discovered by GCD–2s are close to the frequencies used to generate the data.[1] Let us turn to the finite MKL methods. As can be seen from the misclassification error results in Figure 3.3, in this example, the chosen discretization accuracy is insufficient. Although it would be easy to increase the discretization accuracy to improve the results of finite MKL methods,[2] the point is that if a high resolution is needed in a one-dimensional problem, then these methods are likely to face serious difficulties in problems when the space of kernels is more complex, e.g., when the parameterization is multi-dimensional. Note that we are not suggesting that the methods which require discretization are universally inferior, but merely wish to point out that an "appropriate discrete kernel set" might not always be available, nor simply obtained by discretization.

[1]A similar plot for CKL showed that this method also was able to find the key frequencies.
[2]Further experimentation found that a discretization below 0.1 is necessary in this example.

3.2 Summary

In this chapter, we showed, by means of an example, that finite MKL methods might fail when the key kernels are not in the base kernel set. If, for example, base kernels are parameterized from a continuous space, one may have to discretize the space into very fine grids to achieve good performance. This leads to an exponential growth in the number of kernels as a function of the kernel space dimensionality. We will address this problem in Chapters 4 and 5.

Chapter 4

Stochastic Gradient Methods for Multiple Kernel Learning

Stochastic gradient methods are a family of stochastic optimization algorithms that have been used to solve large scale optimization problems. We will show that these algorithms are specifically useful for multiple kernel learning when the goal is to combine many kernels. In this chapter we propose new algorithms for one-stage and two-stage MKL that are based on the stochastic gradient method. Section 4.1 briefly reviews the stochastic gradient method. In Section 4.2, we propose a new algorithm for one-stage MKL. Finally, in Section 4.3 we discuss the applicability of the stochastic gradient algorithm for two-stage MKL.

4.1 Stochastic Gradient Descent

Consider the general problem of minimizing a convex function $J(\theta)$. The stochastic gradient descent (SGD) algorithm is a variant of gradient descent that can be used to find the minimizer of $J(\theta)$. This algorithm uses an unbiased estimate of the gradient instead of the full gradient vector:

$$\theta^{(k)} \leftarrow \theta^{(k-1)} - \eta_k \hat{g}_k,$$

where $\eta_k > 0$ is a learning rate parameter, and $\hat{g}_k$ is an unbiased estimate of the gradient of J at $\theta^{(k-1)}$. When the amount of training data is large, the gradient estimate can be constructed by evaluating the gradient of the loss function over a subset of data, and in the extreme case, on a single example. This often results in a drastic improvement in the per-step performance of the algorithm. Furthermore, it has been shown that the overall runtime of stochastic gradient descent does not depend on the size of training set and sometimes even decreases as the number of training data increases (Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008). Another way to construct the gradient estimate vector, which is our focus in this chapter, is to have a small subset of non-zero coordinates. While it introduces some variance in the gradient estimate, the approach results in an efficient algorithm for high dimensional optimization problems. In the next section, we introduce a stochastic version of the mirror descent algorithm, which is a more general variant of stochastic gradient descent.

4.1.1 A stochastic mirror descent algorithm

Before presenting the algorithm, we begin with a few definitions. Let $d = |\mathcal{I}|$, and $A \subset \mathbb{R}^d$ be nonempty with a convex interior A°. We call the function $\Psi : A \to \mathbb{R}$ a *Legendre* (or barrier) *potential* if it is strictly convex, its partial derivatives exist and are continuous, and for every sequence $\{x_k\} \subset A$ approaching the boundary of A,

$$\lim_{k\to\infty} \|\nabla\Psi(x_k)\| = \infty.$$

Here ∇ is the gradient operator: $\nabla\Psi(x) = (\frac{\partial}{\partial x}\Psi(x))^\top$ is the gradient of Ψ. When ∇ is applied to a non-smooth convex function $J(\theta)$ (J may be such without additional assumptions) then $\nabla J(\theta)$ is defined as any subgradient of J at θ. The corresponding Bregman-divergence $D_\Psi : A \times A^\circ \to \mathbb{R}$ is defined as

$$D_\Psi(\theta, \theta') = \Psi(\theta) - \Psi(\theta') - \langle \nabla\Psi(\theta'), \theta - \theta' \rangle.$$

The Bregman projection $\Pi_{\Psi,K} : A^\circ \to K$ corresponding to the Legendre potential Ψ and a closed convex set $K \subset \mathbb{R}^d$ such that $K \cap A \neq \emptyset$ is defined, for all $\theta \in A^\circ$ as

$$\Pi_{\Psi,K}(\theta) = \operatorname*{arg\,min}_{\theta' \in K\cap A} D_\Psi(\theta', \theta).$$

Algorithm 1 shows the stochastic version of the *mirror descent algorithm* (Nemirovski and Yudin, 1998)[1], with an unbiased gradient estimate. By assumption, $\eta_k > 0$ is deterministic. Note that step 13 of the algorithm is well-defined since $\tilde{\theta}^{(k)} \in A^\circ$ by the assumption that $\|\nabla\Psi(x)\|$ tends to infinity as x approaches the boundary of A. Using the standard proof technique of analyzing the mirror descent algorithm (see, e.g., (Beck and Teboulle, 2003)), the performance of Algorithm 1 can be bounded as follows (the proof is included in Appendix A):

[1]The mirror descent algorithm is a special case of the proximal point algorithm (Rockafellar, 1976).

Algorithm 1 Stochastic mirror descent algorithm

```
1: **Input:**
2: $A, K \subset \mathbb{R}^d$, where $K$ is closed and convex with $K \cap A \neq \emptyset$,
3: $\Psi : A \to \mathbb{R}$ Legendre,
4: Step sizes: $\{\eta_k\}$,
5: A subroutine, GradSampler, to sample the gradient of $J$ at an arbitrary vector
   $\theta \geq 0$
6: **Initialization:**
7: $\theta^{(0)} = \arg\min_{\theta \in K \cap A} \Psi(\theta)$
8: $k = 0$
9: **repeat**
10:     $k = k + 1$
11:     Obtain $\hat{g}_k =$ GradSampler($\theta^{(k-1)}$)
12:     $\tilde{\theta}^{(k)} = \arg\min_{\theta \in A} \left\{ \eta_{k-1} \langle \hat{g}_k, \theta \rangle + D_\Psi(\theta, \theta^{(k-1)}) \right\}$
13:     $\theta^{(k)} = \Pi_{\Psi,K}(\tilde{\theta}^{(k)})$
14: **until** convergence
```

Theorem 4.1. *Assume that Ψ is α-strongly convex with respect to some norm $\|\cdot\|$ (with dual norm $\|\cdot\|_*$) for some $\alpha > 0$, that is, for any $\theta \in A^\circ, \theta' \in A$*

$$\Psi(\theta') - \Psi(\theta) \geq \left\langle \nabla\Psi(\theta), \theta' - \theta \right\rangle + \tfrac{\alpha}{2}\|\theta' - \theta\|^2. \tag{4.1}$$

Suppose, furthermore, that Algorithm 1 is run for T time steps. For $0 \leq k \leq T-1$ let $\mathcal{F}_k$ denote the σ-algebra generated by $\theta_1, \ldots, \theta_k$. Assume that, for all $1 \leq k \leq T$, $\hat{g}_k \in \mathbb{R}^d$ is an unbiased estimate of $\nabla J(\theta^{(k-1)})$ given $\mathcal{F}_{k-1}$, that is,

$$\mathbb{E}\left[\hat{g}_k \middle| \mathcal{F}_{k-1}\right] = \nabla J(\theta^{(k-1)}). \tag{4.2}$$

Further, assume that there exists a deterministic constant $B \geq 0$ such that for all $1 \leq k \leq T$,

$$\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1}\right] \leq B \quad a.s. \tag{4.3}$$

Finally, assume that $\delta = \sup_{\theta' \in K \cap A} \Psi(\theta') - \Psi(\theta^{(0)})$ is finite. Then, if $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{BT}}$ for all $k \geq 1$, it holds that

$$\mathbb{E}\left[J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right)\right] - \inf_{\theta \in K \cap A} J(\theta) \leq \sqrt{\frac{2B\delta}{\alpha T}}. \tag{4.4}$$

Furthermore, if

$$\|\hat{g}_k\|_*^2 \leq B' \quad a.s. \tag{4.5}$$

for some deterministic constant B' and $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{B'T}}$ for all $k \geq 1$ then, for any $0 < \varepsilon < 1$, it holds with probability at least $1 - \varepsilon$ that

$$J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right) - \inf_{\theta \in K \cap A} J(\theta) \leq \sqrt{\frac{2B'\delta}{\alpha T}} + 4\sqrt{\frac{B'\delta \log\frac{1}{\varepsilon}}{\alpha T}}. \tag{4.6}$$

The convergence rate in the above theorem can be improved if stronger assumptions are made on J, for example if J is assumed to be strongly convex, see, for example, Hazan et al. (2007); Hazan and Kale (2011).

Efficient implementation of Algorithm 1 depends on efficient implementations of steps 11-13, namely, computing an estimate of the gradient, solving the minimization for $\tilde{\theta}^{(k)}$, and projecting it onto K. The first problem is related to the choice of the gradient estimate we use, which, in turn, depends on the structure of the feature space, while the last two problems depend on the choice of the Legendre function. In the next sections, we examine how these choices can be made to get a practical variant of the algorithm for the MKL problem.

4.2 Stochastic Gradient Descent for One-Stage MKL

We aim to solve the optimization problem 2.9 of one-stage MKL using the stochastic gradient approach. When $\mathcal{I}$, the kernel index set, is small, or moderate in size, the joint-convexity of J allows one to use off-the-shelf solvers to find the joint minimum of J. However, when $\mathcal{I}$ is large, off-the-shelf solvers might be slow or they may run out of memory. Targeting this situation we propose the following approach: Exploiting again that $J(w, \theta)$ is jointly convex in (w, θ), find the optimal weights by finding the minimizer of

$$J(\theta) \stackrel{\text{def}}{=} \inf_{w} J(w, \theta), \tag{4.7}$$

or, alternatively,

$$J(\theta) = J(w^*(\theta), \theta),$$

where

$$w^*(\theta) \stackrel{\text{def}}{=} \arg\min_{w} J(w, \theta).$$

Here we have slightly abused notation by reusing the symbol J. Note that $J(\theta)$ is convex by the joint convexity of $J(w, \theta)$ (see, e.g., Boyd and Vandenberghe, 2004, Section 3.2.5). Also, note that $w^*(\theta)$ exists and is well-defined since the minimizer of $J(\cdot, \theta)$ is unique for any $\theta \in \Delta_\nu$ (see also Proposition 4.2 below). Again, exploiting the joint convexity of $J(w, \theta)$, we find that if θ^* is the minimizer of $J(\theta)$, then $w^*(\theta^*)$ will be an optimal solution to the original problem (2.8). To optimize $J(\theta)$ we propose to use stochastic gradient descent with artificially injected randomness to avoid the need to fully evaluate the gradient of J. More precisely, our proposed algorithm is an instance of the stochastic mirror descent algorithm shown in Algorithm 1 where in each time step only one coordinate of the gradient is sampled.

In order to apply Algorithm 1 to the one-stage MKL problem we must specify the gradient estimates $\hat{g}_k$. We start by considering importance sampling based estimates. First, however, let us verify whether the gradient exists. Along the way, we will also derive some explicit expressions which will help us later.

Closed-form expressions for the gradient

Let us first consider how $w^*(\theta)$ can be calculated for a fixed value of θ. As it will turn out, this calculation will be useful not only when the procedure is stopped (to construct the predictor $f_{w^*(\theta)}$), but also during the iterations when we will need to calculate the derivative of J with respect to θ_i. The following proposition summarizes how $w^*(\theta)$ can be obtained. Note that this type of result is standard (see, e.g., Shawe-Taylor and Cristianini, 2004; Schölkopf and Smola, 2002), thus we include it only for the sake of completeness (the proof is included in Appendix A).

Proposition 4.2. *For $t \in \{1, \dots, n\}$, let $\ell_t^* : \mathbb{R} \to \mathbb{R}$ denote the convex conjugate of ℓ_t: $\ell_t^*(v) = \sup_{\tau \in \mathbb{R}} \{v\tau - \ell_t(\tau)\}$, $v \in \mathbb{R}$. For $i \in \mathcal{I}$, recall that $\kappa_i(x, x') = \langle \phi_i(x), \phi_i(x') \rangle$, and let $\mathbf{K}_i = (\kappa_i(x_t, x_s))_{t,s \in \{1,\dots,n\}}$ be the $n \times n$ kernel matrix underlying κ_i and let $\mathbf{K}_\theta = \sum_{i \in \mathcal{I}} \frac{\theta_i}{\rho_i^2} \mathbf{K}_i$ be the kernel matrix underlying $\kappa_\theta = \sum_{i \in \mathcal{I}} \frac{\theta_i}{\rho_i^2} \kappa_i$. Then, for any fixed θ, the minimizer $w^*(\theta)$ of $J(\cdot, \theta)$ satisfies*

$$w_i^*(\theta) = \frac{\theta_i}{\rho_i^2} \sum_{t=1}^{n} \alpha_t^*(\theta) \phi_i(x_t), \quad i \in \mathcal{I}, \tag{4.8}$$

where

$$\alpha^*(\theta) = \operatorname*{arg\,min}_{\alpha \in \mathbb{R}^n} \left\{ \frac{1}{2} \alpha^\top \mathbf{K}_\theta \alpha + \frac{1}{n} \sum_{t=1}^{n} \ell_t^*(-n\alpha_t) \right\}. \tag{4.9}$$

Based on this proposition, we can compute the predictor $f_{w^*(\theta)}$ using the kernels $\{\kappa_i\}_{i \in \mathcal{I}}$ and the dual variables $(\alpha_t^*(\theta))_{t \in \{1,\dots,n\}}$:

$$f_{w^*(\theta)}(x) = \sum_{i \in \mathcal{I}} \langle w_i^*(\theta), \phi_i(x) \rangle = \sum_{t=1}^{n} \alpha_t^*(\theta) \kappa_\theta(x_t, x).$$

Calculating the derivative of $J(\theta)$

In this section we show that under mild conditions the derivative of J exists and we also give explicit forms. These derivations are quite standard and a similar argument can be found in, e.g., the paper by Rakotomamonjy et al. (2008) specialized to the case when ℓ_t is the hinge loss.

As it is well-known, thanks to the implicit function theorem (e.g., Brown and Page, 1970, Theorem 7.5.6), provided that $J = J(w,\theta)$ is such that $\frac{\partial^2}{\partial\theta\partial w}J(w,\theta)$ and $\frac{\partial}{\partial w}J(w,\theta)$ are continuous, the gradient of $J(\theta)$ can be computed by evaluating the partial derivative of $J(w,\theta)$ with respect to θ at $(w^*(\theta),\theta))$, that is,

$$\partial_\theta J(\theta) = \frac{\partial}{\partial\theta}\, J(w,\theta)|_{w=w^*(\theta)}\,.$$

Note that the derivative is well-defined only if $\theta > 0$, that is, when no coordinate of θ is zero, in which case

$$\frac{\partial}{\partial\theta}J(w^*(\theta),\theta) = -\frac{1}{2}\left(\frac{\rho_i^2\|w_i^*(\theta)\|_2^2}{\theta_i^2}\right)_{i\in\mathcal{I}}. \tag{4.10}$$

If $\theta_i = 0$ for some $i \in \mathcal{I}$, we define the derivative in a continuous manner as

$$\frac{\partial}{\partial\theta}J(\theta) = \lim_{\substack{\theta'\to\theta\\ \theta'\in\Delta,\theta'>0}} \frac{\partial}{\partial\theta}J(\theta') \tag{4.11}$$

assuming that the limit exists. From (4.8) we get, for any $i \in \mathcal{I}$,

$$\|w_i^*(\theta)\|_2^2 = \frac{\theta_i^2}{\rho_i^4}\alpha^*(\theta)^\top \mathbf{K}_i\alpha^*(\theta).$$

Combining with (4.10) we obtain

$$\frac{\partial}{\partial\theta}J(\theta) = \frac{\partial}{\partial\theta}J(w^*(\theta),\theta) = -\frac{1}{2}\left(\frac{\alpha^*(\theta)^\top\mathbf{K}_i\alpha^*(\theta)}{\rho_i^2}\right)_{i\in\mathcal{I}}. \tag{4.12}$$

Now, by (4.11) and the implicit function theorem, $\alpha^*(\theta)$ is a continuous function of θ provided that the functions ℓ_t^*, $t \in \{1,\ldots,n\}$, are twice continuously differentiable. This shows that under the conditions listed so far, the limit in (4.11) exists. In the application we shall be concerned with, these conditions can be readily verified.

Importance sampling based estimates

Let $d = |\mathcal{I}|$ and let e_i, $i \in \mathcal{I}$, denote the i^{th} unit vector of the standard basis of $\mathbb{R}^d$, that is, the i^{th} coordinate of e_i is 1 while the others are 0. Introduce

$$g_{k,i} = \left\langle \nabla J(\theta^{(k-1)}), e_i \right\rangle, \quad i \in \mathcal{I} \tag{4.13}$$

to denote the i^{th} component of the gradient of J in iteration k (that is, $g_{k,i}$ can be computed based on (4.12)). Let $s_{k-1} \in [0,1]^{\mathcal{I}}$ be a distribution over $\mathcal{I}$, computed in some way based on the information available up to the end of iteration $k-1$ of the algorithm (formally, s_{k-1} is $\mathcal{F}_{k-1}$-measurable). Define the importance sampling based gradient estimate to be

$$\hat{g}_{k,i} = \frac{\mathbb{I}_{\{I_k=i\}}}{s_{k-1,I_k}} g_{k,I_k}, \quad i \in \mathcal{I}, \qquad \text{where } I_k \sim s_{k-1,\cdot}\ . \tag{4.14}$$

That is, the gradient estimate is obtained by first sampling an index from $s_{k-1,\cdot}$ and then setting the gradient estimate to be zero at all indices $i \in \mathcal{I}$ except when $i = I_k$, in which case its value is set to be the ratio

$$\frac{g_{k,I_k}}{s_{k-1,I_k}}.$$

It is easy to see that as long as $s_{k-1,i} > 0$ holds whenever $g_{k,i} \neq 0$, then it holds that

$$\mathbb{E}\left[\hat{g}_k \middle| \mathcal{F}_{k-1}\right] = \nabla J(\theta^{(k-1)}) \quad \text{a.s.}$$

Let us now derive the conditions under which the second moment of the gradient estimate stays bounded. Define

$$C_{k-1} = \left\| \nabla J(\theta^{(k-1)}) \right\|_1 .$$

Given the expression for the gradient of J shown in (4.12), we see that

$$\sup_{k \geq 1} \ C_{k-1} < \infty$$

will always hold provided that $\alpha^*(\theta)$ is continuous since $(\theta^{(k-1)})_{k\geq 1}$ is guaranteed to belong to a compact set (the continuity of α^* was discussed earlier in this section). Define the probability distribution $q_{k-1,\cdot}$ as follows:

$$q_{k-1,i} = \frac{1}{C_{k-1}} \left| g_{k,i} \right| , \quad i \in \mathcal{I}. \tag{4.15}$$

Then it holds that

$$\begin{aligned} \|\hat{g}_k\|_*^2 &= \frac{1}{s_{k-1,I_k}^2} g_{k,I_k}^2 \|e_{I_k}\|_*^2 \\ &= \frac{q_{k-1,I_k}^2}{s_{k-1,I_k}^2} C_{k-1}^2 \|e_{I_k}\|_*^2. \end{aligned}$$

Therefore, it also holds that

$$\begin{aligned} \mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1} \right] &= C_{k-1}^2 \sum_{i \in \mathcal{I}} \frac{q_{k-1,i}^2}{s_{k-1,i}} \|e_i\|_*^2 \\ &\leq C_{k-1}^2 \max_{i \in \mathcal{I}} \frac{q_{k-1,i}}{s_{k-1,i}} \|e_i\|_*^2. \end{aligned}$$

This shows that

$$\sup_{k\geq 1} \mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1} \right] < \infty$$

will hold as long as

$$\sup_{k\geq 1} \max_{i\in\mathcal{I}} \frac{q_{k-1,i}}{s_{k-1,i}} < \infty \tag{4.16}$$

and

$$\sup_{k\geq 1} C_{k-1} < \infty. \tag{4.17}$$

Note that when $s_{k-1} = q_{k-1}$, the gradient estimate becomes

$$\hat{g}_{k,i} = C_{k-1}\mathbb{I}_{\{I_t = i\}}.$$

That is, in this case we see that in order to be able to calculate $\hat{g}_{k,i}$, we need to be able to calculate C_{k-1} efficiently.

Choosing the potential Ψ

The efficient sampling of the gradient is not the only practical issue, since the choice of the Legendre function and the convex set K may also cause some complications. For example, if $\Psi(x) = \sum_{i\in\mathcal{I}} x_i(\ln x_i - 1)$, then the resulting algorithm is exponential weighting, and one needs to store and update $|\mathcal{I}|$ weights, which is clearly infeasible if $|\mathcal{I}|$ is very large (or infinite). On the other hand, if $\Psi(x) = \frac{1}{2}\|x\|_2^2$ and we project onto $K = \Delta_2$, the positive quadrant of the ℓ^2-ball (with $A = [0,\infty)^{\mathcal{I}}$), we obtain a stochastic projected gradient method, shown in Algorithm 2. This is in fact the algorithm that we use in the experiments. Note that in (2.8) this corresponds to using $p = 4/3$. The reason we made this choice is because in this case projection is a simple scaling operation. We observed in practice that had we chosen $K = \Delta_1$, the ℓ^2-projection would very often cancel many of the nonzero components, resulting in an overall slow progress. Based on the above calculations and Theorem 4.1 we obtain the following performance bound for our algorithm.

Corollary 4.3. *Assume that $\alpha^*(\theta)$ is continuous on Δ_2. Then there exists a $C > 0$ such that $\|\frac{\partial}{\partial\theta}J(\theta)\|_1 \leq C$ for all $\theta \in \Delta_2$. Let $B = \frac{1}{2}C^2 \max_{i\in\mathcal{I}, 1\leq k\leq T} \frac{q_{k-1,i}}{s_{k-1,i}}$. If Algorithm 2 is run for T steps with $\eta_{k-1} = \eta = 1/\sqrt{BT}, k = 1,\ldots,T$, then, for all $\theta \in \Delta_2$,*

$$\mathbb{E}\left[J\left(\frac{1}{T}\sum_{k=1}^{T} \theta^{(k-1)} \right) \right] - J(\theta) \leq \sqrt{\frac{B}{T}}.$$

Algorithm 2 Projected stochastic gradient algorithm.

```
1: **Inputs:**
2: Step sizes: $\{\eta_k\}$
3: **Initialization:**
4: $\Psi(x) = \frac{1}{2}\|x\|_2^2$,
5: $\theta^{(0)} = 0$,
6: $k = 0$,
7: **repeat**
8:     $k = k + 1$
9:     Sample a gradient estimate $\hat{g}_k$ of $g(\theta^{(k-1)})$ randomly according to (4.14)
10:    $\theta^{(k)} = \Pi_{\Psi,\Delta_2}(\theta^{(k-1)} - \eta_{k-1}\hat{g}_k)$
11: **until** convergence
```

Note that to implement Algorithm 2 efficiently, one has to be able to sample from $s_{k-1,\cdot}$ and compute the importance sampling ratio $g_{k,i}/s_{k,i}$ efficiently for any k and i.

4.2.1 Example: Learning polynomial kernels

In this section we show how our method can be applied in the context of one-stage multiple kernel learning to an important family of kernels. We provide an example when the kernels in $\mathcal{I}$ are tensor products of a set of base kernels (we call this learning polynomial kernels).

Assume that we are given a set of base kernels $\{\kappa_1, \ldots, \kappa_r\}$. In this section we consider the set $\mathcal{K}_D$ of product kernels of degree at most D: Choose

$$\mathcal{I} = \{(r_1, \ldots, r_d) \,:\, 0 \le d \le D, 1 \le r_i \le r\}$$

and the multi-index $r_{1:d} = (r_1, \ldots, r_d) \in \mathcal{I}$ defines the kernel

$$\kappa_{r_{1:d}}(x, x') = \prod_{i=1}^{d} \kappa_{r_i}(x, x').$$

For $d = 0$ we define $\kappa_{r_{1:0}}(x, x') = 1$. Note that indices that are the permutations of each other define the same kernel. In the language of statistical modeling, $\kappa_{r_{1:d}}$ models interactions of order d between the features underlying the base kernels $\kappa_1, \ldots, \kappa_r$. Also note that $|\mathcal{I}| = \Theta(r^D)$, that is, the cardinality of $\mathcal{I}$ grows exponentially fast in D.

We assume that $\rho_{r_{1:d}}$ depends only on d, the order of interactions in $\kappa_{r_{1:d}}$. By abusing notation, we will write ρ_d in the rest of this section to emphasize this.[2] Our

[2]Using importance sampling, more general weights can also be accommodated too, without affecting the results as long as the range of weights $(\rho_{r_{1:d}})$ is kept under control for all d.

proposed sampling procedure to sample from $q_{k-1,\cdot}$ is shown in Algorithm 3. The algorithm is written to return a multi-index $(z_1, \ldots, z_d)$ that is drawn from $q_{k-1,\cdot}$. The key idea underlying the algorithm is to exploit that

$$\left(\sum_{j=1}^{r} \kappa_j\right)^d = \sum_{r_{1:d} \in \mathcal{I}} \kappa_{r_{1:d}}.$$

Note that in the description of the algorithm $\odot$ denotes the matrix entrywise product (a.k.a. Schur, or Hadamard product) and $A^{\odot s}$ denotes $\underbrace{A \odot \ldots \odot A}_{s}$, and we set the priority of $\odot$ to be higher than that of the ordinary matrix product.

Let us now discuss the complexity of Algorithm 3. For this, first note that computing all the Hadamard products $S^{\odot d'}, d' = 0, \ldots, D$ requires $O(Dn^2)$ computations. Multiplication with M_{k-1} can be done in $O(n^2)$. Finally, note that each iteration of the for loop takes $O(rn^2)$ steps, which results in the overall worst-case complexity of $O(rn^2 D)$ if $\alpha^*(\theta^{(k-1)})$ is readily available. The computational complexity of determining $\alpha^*(\theta^{(k-1)})$ depends on the exact form of ℓ_t, and can be done efficiently in many situations: if, for example, ℓ_t is the squared error, then α^* can be computed in $O(n^3)$ time. An obvious improvement to the approach described here, however, would be to subsample the empirical loss, Loss_n, which can bring further computational improvements. However, the exploration of this is left for future work.

Finally, note that despite the exponential cardinality of $\mathcal{I}$, due to the strong algebraic structure of the space of kernels, C_{k-1} can be calculated efficiently. In fact, it is not hard to see that with the notation of the algorithm,

$$C_{k-1} = \sum_{d'=0}^{D} \delta(d').$$

This also shows that if ρ_d decays "fast enough", C_{k-1} can be bounded independently of the cardinality of $\mathcal{I}$. Next, we show the correctness of Algorithm 3.

Correctness of the sampling procedure

In this section we prove the correctness of Algorithm 3. As said earlier, we assume that $\rho_{r_{1:d}}$ depends only on d, the order of interactions in $\kappa_{r_{1:d}}$ and, by abusing notation, we will write ρ_d to emphasize this. Let us now consider how one can sample from $q_{k-1,\cdot}$. The implementation relies on the fact that $(\sum_{j=1}^{r} \kappa_j)^d = \sum_{r_{1:d} \in \mathcal{I}} \kappa_{r_{1:d}}$.

Algorithm 3 Polynomial kernel sampling. The symbol $\odot$ denotes the Hadamard product/power.

```
1: Input:
2: $\alpha \in \mathbb{R}^n$, the solution to the dual problem;
3: kernel matrices $\{\mathbf{K}_1, \dots, \mathbf{K}_r\}$;
4: the degree $D$ of the polynomial kernel,
5: the weights $(\rho_0^2, \dots, \rho_D^2)$.
6: $S \leftarrow \sum_{j=1}^r \mathbf{K}_j$, $M \leftarrow \alpha\alpha^\top$
7: $\delta(d') \leftarrow \rho_{d'}^{-2} \left\langle M, S^{\odot d'} \right\rangle, \quad d' \in \{0, \dots, D\}$
8: Sample $d$ from $\delta(\cdot) / \sum_{d'=0}^D \delta(d')$
9: for $i = 1$ to $d$ do
10:     $\pi(j) \leftarrow \frac{\mathrm{tr}(M\, S^{\odot(d-i)} \odot \mathbf{K}_j)}{\mathrm{tr}(M\, S^{\odot(d-i+1)})}, \quad j \in \{1, \dots, r\}$
11:     Sample $z_i$ from $\pi(\cdot)$
12:     $M \leftarrow M \odot \mathbf{K}_{z_i}$
13: end for
14: return $(z_1, \dots, z_d)$
```

Remember that we denoted the kernel matrix underlying some kernel κ by $\mathbf{K}_\kappa$, and recall that $\mathbf{K}_\kappa$ is an $n \times n$ matrix. For brevity, in the rest of this section for $\kappa = \kappa_{r_{1:d}}$ we will write $\mathbf{K}_{r_{1:d}}$ instead of $\mathbf{K}_{\kappa_{r_{1:d}}}$. Define $M_{k-1} = \alpha^*(\theta^{(k-1)})\alpha^*(\theta^{(k-1)})^\top$. using (4.12) and the rotation property of trace, we have

$$g_{k,r_{1:d}} = -\rho_d^{-2} \mathrm{tr}(M_{k-1}\, \mathbf{K}_{r_{1:d}})\,. \tag{4.18}$$

The plan to sample from

$$q_{k-1,\cdot} = \frac{|g_{k,\cdot}|}{\sum_{r_{1:d} \in \mathcal{I}} |g_{k,r_{1:d}}|}$$

is as follows: We first draw the order of interactions, $\hat{d} \in \{0, \dots, D\}$. Then, given $\hat{d} = d$, we restrict the draw of the random multi-index $R_{1:d}$ to the set $\{r_{1:d} \in \mathcal{I}\}$. A multi-index will be sampled in a $\hat{d}$-step process: in each step we will randomly choose an index from the indices of base kernels according to the following distributions. Let $S = \mathbf{K}_1 + \ldots + \mathbf{K}_r$, and let

$$\mathbb{P}\left(\hat{d} = d | \mathcal{F}_{k-1}\right) = \frac{\rho_d^{-2} \mathrm{tr}(M_{k-1} S^{\odot d})}{\sum_{d'=0}^D \rho_{d'}^{-2} \mathrm{tr}(M_{k-1} S^{\odot d'})}$$

and

$$\begin{aligned} &\mathbb{P}\left(R_1 = r_1 | \mathcal{F}_{k-1}, \hat{d} = d\right) \\ &\quad = \frac{\mathrm{tr}(M_{k-1}\, \mathbf{K}_{r_1} \odot S^{\odot(d-1)})}{\sum_{r_1'=1}^r \mathrm{tr}(M_{k-1}\, \mathbf{K}_{r_1'} \odot S^{\odot(d-1)})}\,, \end{aligned}$$

$$\begin{aligned}
&\mathbb{P}\left(R_2 = r_2|\mathcal{F}_{k-1}, \hat{d} = d, R_1 = r_1\right) \\
&\quad = \frac{\operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \mathbf{K}_{r_2} \odot S^{\odot(d-2)})}{\sum_{r_2'=1}^{r} \operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \mathbf{K}_{r_2'} \odot S^{\odot(d-2)})}\,, \\
&\quad \vdots \\
&\mathbb{P}\left(R_d = r_d|\mathcal{F}_{k-1}, \hat{d} = d, R_{1:d-1} = r_{1:d-1}\right) \\
&\quad = \frac{\operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{d-1}} \odot \mathbf{K}_{r_d})}{\sum_{r_d'=1}^{r} \operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{d-1}} \odot \mathbf{K}_{r_d'})}\,,
\end{aligned}$$

where we used the sequence notation, i.e., $r_{1:p}$ denotes the sequence $(r_1, \ldots, r_p)$. We have, by the linearity of trace and the definition of S that

$$\begin{aligned}
&\sum_{r_1'=1}^{r} \operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1'} \odot S^{\odot(d-1)}) \\
&\qquad = \operatorname{tr}\Big(M_{k-1}\,\big(\sum_{r_1'=1}^{r} \mathbf{K}_{r_1'}\big) \odot S^{\odot(d-1)}\Big) \\
&\qquad = \operatorname{tr}(M_{k-1}\,S \odot S^{\odot(d-1)}) \\
&\qquad = \operatorname{tr}(M_{k-1}\,S^{\odot d}).
\end{aligned}$$

Similarly,

$$\begin{aligned}
&\sum_{r_j'=1}^{r} \operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{j-1}} \odot \mathbf{K}_{r_j'} \odot S^{\odot(d-j)}) \\
&\quad = \operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{j-1}} S^{\odot(d-j+1)})\,.
\end{aligned}$$

Thus, as desired,

$$\begin{aligned}
&\mathbb{P}\left(\hat{d} = d, R_{1:d} = r_{1:d}|\mathcal{F}_{k-1}\right) \\
&\quad = \mathbb{P}\left(R_{1:d} = r_{1:d}|\mathcal{F}_{k-1}, \hat{d} = d\right)\mathbb{P}\left(\hat{d} = d|\mathcal{F}_{k-1}\right) \\
&\quad = \frac{\operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{d-1}} \odot \mathbf{K}_{r_d})}{\operatorname{tr}(M_{k-1}\,S^{\odot d})} \\
&\qquad\qquad \times \frac{\rho_d^{-2}\operatorname{tr}(M_{k-1} S^{\odot d})}{\sum_{d'=0}^{D} \rho_{d'}^{-2}\operatorname{tr}(M_{k-1} S^{\odot d'})} \\
&\quad = \frac{\rho_d^{-2}\operatorname{tr}(M_{k-1}\,\mathbf{K}_{r_1} \odot \ldots \odot \mathbf{K}_{r_{d-1}} \odot \mathbf{K}_{r_d})}{\sum_{d'=0}^{D} \rho_{d'}^{-2}\operatorname{tr}(M_{k-1} S^{\odot d'})}.
\end{aligned}$$

An optimized implementation of drawing these random variables is shown as Algorithm 3. The algorithm is written to return the multi-index $R_{1:d}$, and the random kernel matrix $M = \mathbf{K}_{R_1} \odot \ldots \odot \mathbf{K}_{R_d}$ is also available.

4.2.2 Example: Learning Gaussian kernels

In this section we explore the possibility of applying our stochastic gradient method to another important family of kernels that include uncountably many kernels: parameterized Gaussian kernels. We consider an index set of the form

$$\mathcal{I} = \{\sigma \, : \, \sigma \in \Sigma\} \, ,$$

where Σ is the parameter space and is assumed to be bounded. We consider two categories of Gaussian kernels based on the dimensionality of the kernel parameter. In the first category each kernel is parameterized by a single positive real number, σ:

$$\kappa_\sigma(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\sigma^2}\right) . \tag{4.19}$$

In the second category kernels are parameterized by a diagonal covariance matrix, i.e. there is one parameter per input variable:

$$\kappa_\sigma(x, x') = \exp\left(-\sum_{i=1}^{r} \frac{(x^{(i)} - x'^{(i)})^2}{\sigma_i^2}\right) . \tag{4.20}$$

In this section we explain how our stochastic mirror descent algorithm can be applied to continuously-parameterized Gaussian kernels. Similar to (4.18) the coordinates of gradient vector are given by

$$g_{k,\sigma} = -\rho_\sigma^{-2} \mathrm{tr}(M_{k-1} \, \mathbf{K}_\sigma) \, ,$$

where, for simplicity of notation, $\mathbf{K}_{\kappa_\sigma}$ is denoted by $\mathbf{K}_\sigma$. We aim to sample from

$$q_{k-1,\cdot} = \frac{1}{S_{k-1}} |g_{k,\cdot}|,$$

where

$$S_{k-1} = \int_{\sigma' \in \Sigma} |g_{k,\sigma'}| \; \mathrm{d}\sigma'$$

is a normalization factor. For simplicity, we assume that $\rho_\sigma = 1$, $\sigma \in \Sigma$.

It is not easy to construct the density function $q_{k-1,\cdot}$ and directly sample from it. Therefore, in this case, we resort to sampling from an auxiliary distribution. In particular, we use importance sampling and propose to sample from an auxiliary density function over Σ. Let σ_k denote the kernel parameter sampled at iteration k from an auxiliary distribution defined by a density function denoted by $s_{k-1,\cdot}$. Similar to (4.14) an unbiased estimate of gradient in this case is constructed by

$$\hat{g}_{k,\sigma} = \frac{\delta(\sigma - \sigma_k)}{s_{k-1,\sigma_k}} \mathrm{tr}(M_{k-1} \mathbf{K}_{\sigma_k}), \quad \sigma \in \Sigma, \quad \text{where } \sigma_k \sim s_{k-1,\cdot}, \tag{4.21}$$

where $\delta(\cdot)$ is the Dirac delta function. It is straightforward to compute $\mathrm{tr}(M_{k-1}\mathbf{K}_{\sigma_k})$. It remains to specify s_{k-1,σ_k}. There are many choices of $s_{k-1,\cdot}$ that result in different values of variance of the gradient estimate. Next, we propose examples of $s_{k-1,\cdot}$ that can be used for learning one- and multi-dimensional Gaussian kernels.

Sampling Gaussian kernels with one-dimensional parameter

We propose the following sampling procedure when the Gaussian kernel parameter is one-dimensional. We begin by solving the following optimization problem:

$$\sigma^* = \arg\max_{\sigma\in\Sigma} \ \mathrm{tr}(M_{k-1}\mathbf{K}_\sigma), \tag{4.22}$$

where $\mathbf{K}_\sigma$ is the kernel matrix corresponding to kernel function (4.19). This optimization problem has been considered by previous approaches that deal with continuously-parameterized kernels (Argyriou et al., 2005; Gehler and Nowozin, 2008). When kernel parameter, σ, is the bandwidth of a Gaussian kernel, problem (4.22) is not convex. However, as noted in, e.g., Argyriou et al. (2005) and Argyriou et al. (2006) there are usually very few local optima (less than 5). Once σ^* is determined, we use a truncated normal density function as the sampling distribution, $s_{k-1,\cdot}$. The truncated normal distribution is limited to Σ. We set its mean to σ^* and set an arbitrary variance.[3] Note that for a truncated normal distribution, limited to a bounded Σ, we have

$$s_{k-1,\sigma} > 0, \quad k \geq 1, \ \sigma \in \Sigma.$$

This means that the bounded variance conditions (4.16) and (4.17) are satisfied. Let σ_k be the Gaussian kernel parameter sampled from $s_{k-1,\cdot}$. To construct an unbiased estimate of gradient we need to compute s_{k-1,σ_k}. This value can be computed for one-dimensional truncated normal distribution. To see this let $\Phi(\cdot)$ denote the cumulative distribution function of the standard normal distribution.[4] The density function of the truncated normal distribution with mean μ and variance σ^2 in interval $[l, u]$ is given by

$$f(x;\mu,\sigma^2,l,u) = \frac{\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)}{\Phi\left(\frac{u-\mu}{\sigma}\right) - \Phi\left(\frac{l-\mu}{\sigma}\right)}.$$

Next, we present a sampling procedure for multi-dimensional Gaussian kernels.

[3] In the experiments, as a heuristic choice, we set the variance to ${\sigma^*}^2$.

[4] The value of $\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}\exp(-t^2/2)\,dt$ can be obtained using math libraries.

Algorithm 4 Multi-dimensional Gaussian kernel sampling. The symbol $\odot$ denotes the Hadamard product.

```
1: Input:
2: $r$: number of dimensions of input variable
3: Initialization:
4: $\alpha \in \mathbb{R}^n$, the solution to the dual problem.
5: $M \leftarrow \alpha\alpha^\top$
6: $P \leftarrow 1$ /* sampling probability */
7: for $i = 1$ to $r$ do
8:     $\sigma^* = \arg\max_{\sigma\in\Sigma} \mathrm{tr}(M\mathbf{K}_\sigma)$, where $\mathbf{K}_\sigma$ is the kernel matrix of a Gaussian
       kernel with one-dimensional parameter $\sigma$.
9:     Let $\pi(\cdot;\sigma^*, s, \Sigma)$ be the density function of a one-dimensional truncated nor-
       mal distribution bounded to interval $\Sigma$, with mean $\sigma^*$, and arbitrary variance
       $s^2$.
10:    Sample $\sigma_i$ from $\pi(\cdot;\sigma^*, s, \Sigma)$
11:    $M \leftarrow M \odot \mathbf{K}_{\sigma_i}$
12:    $P \leftarrow P \cdot \pi(\sigma_i;\sigma^*, s, \Sigma)$
13: end for
14: return $(\sigma_1, \ldots, \sigma_r)$ and $P$
```

Sampling Gaussian kernels with multi-dimensional parameter.

Instead of sampling directly from a multi-dimensional truncated normal distribution, which could be expensive, we resort to a faster iterative algorithm that samples from one-dimensional truncated normal distributions iteratively. Note that the multi-dimensional Gaussian kernels we consider can be expressed as

$$\kappa_\sigma(x, x') = \Pi_{i=1}^r \kappa_{\sigma_i}(x, x'),$$

where $\sigma = (\sigma_i)_{i=1}^r$ is the r-dimensional kernel parameter and κ_{σ_i}'s are one-dimensional Gaussian kernels. The goal is to sample one-dimensional kernel parameters $\sigma_1, \ldots, \sigma_r$ iteratively. To sample each parameter $\sigma^{(i)}$ we use a procedure similar to the method described above for Gaussian kernels with a one-dimensional parameter. Algorithm 4 shows the procedure of multi-dimensional Gaussian kernel sampling. One concern regarding this sampling procedure is that the variance of the gradient estimate might become intractably large as the number of input dimensions increases. We leave further investigation of this issue and possible alternatives to future work.

4.3 Stochastic Gradient Descent for Two-Stage MKL

In this section we explore the possibility of applying the stochastic mirror descent algorithm described in Section 4.1.1 to two-stage MKL. In particular, we focus on

Euclidean distance minimization, which we showed in Section 2.2.2 is equivalent to alignment maximization, as the underlying optimization problem for two-stage MKL. Recall the objective function of two-stage kernel learning given in (2.12). We repeat this optimization problem here:

$$\min_{\theta \geq 0} \qquad J(\theta) = -b^\top \theta + C\theta^\top M \theta,$$

where $b_i = \langle \mathbf{K}_i, \hat{\mathbf{K}}^* \rangle$, and $M_{ij} = \langle \mathbf{K}_i, \mathbf{K}_j \rangle$, $i, j \in \mathcal{I}$. Recall that the value of C plays no role in the alignment rate of convergence. Therefore, for simplicity, we set $C = \frac{1}{2}$. The gradient of J in iteration k is given by:

$$\nabla J(\theta^{(k-1)}) = \left(\left\langle \mathbf{K}_i, \sum_{j \in \mathcal{I}} \theta_j^{(k-1)} \mathbf{K}_j - \hat{\mathbf{K}}^* \right\rangle_F \right)_{i \in \mathcal{I}} . \tag{4.23}$$

Recall that in order to directly sample from $q_{k-1,\cdot}$, defined in (4.15), we need an efficient way of computing C_{k-1}. For the one-stage MKL objective function, discussed in Section 4.2, all coordinates of $-\frac{\partial}{\partial \theta} J(\theta)$ are non-negative. By exploiting this property and that $(\sum_{j=1}^r \kappa_j)^d = \sum_{r_{1:d} \in \mathcal{I}} \kappa_{r_{1:d}}$ we were able to design an efficient sampling procedure for polynomial kernels in the one-stage MKL framework. In the case of Euclidean distance minimization (or alignment maximization), however, this property does not hold. Note that in (4.23) there is no guarantee that all coordinates have the same sign. We leave further investigation of the application of stochastic gradient descent to two-stage MKL to future work.

4.4 Summary

In this chapter we introduced a new method for learning a predictor by combining exponentially or infinitely many linear predictors using a stochastic mirror descent algorithm. In the one-stage MKL framework we derived finite-time performance bounds that show that the method efficiently optimizes our proposed criterion. Our proposed method is a variant of a stochastic gradient descent algorithm, where the main trick is the careful construction of an unbiased randomized estimate of the gradient vector that keeps the variance of the method under control, and can be computed efficiently when the base kernels have a certain special combinatorial structure. We presented an efficient sampling procedure for the practically important problem of learning polynomial kernels. We showed that our method is able to compute an optimal solution in polynomial time as a function of the *logarithm* of the number of base kernels. To our knowledge, our algorithm is the first method for learning kernel combinations that achieve such an exponential reduction in complexity while satisfying strong performance guarantees, thus opening up the option of applying it to an extremely large number of kernels. Furthermore, we proposed efficient algorithms to apply this method to the case when infinitely many kernels are to be combined, such as the case of learning a combination of Gaussian kernels. We also started to explore the application of our new stochastic gradient algorithm to two-stage MKL with Euclidean distance minimization. However, in this case, it is not straightforward to compute an unbiased estimate of the gradient. Further investigation of this case is left to future work. Later, in Chapter 6, we will empirically demonstrate the efficiency of our method in one-stage MKL. We will compare our new algorithm to a representative set of algorithms from the literature on a variety of synthetic and real datasets.

Chapter 5

Greedy Coordinate Descent Methods for Multiple Kernel Learning

In this chapter, we explain how greedy coordinate descent (GCD) can be applied to multiple kernel learning. In particular, GCD is useful when one deals with infinitely-many kernels. In Section 5.1 we review the greedy coordinate descent algorithm. In Section 5.2 we explain how GCD can be applied to one-stage multiple kernel learning. Afterwards, in Section 5.3, we provide the details of applying GCD to two-stage multiple kernel learning.

5.1 Greedy Coordinate Descent

Coordinate descent is a variant of the gradient descent algorithm in which only one coordinate is updated in each iteration. There are different variants of coordinate descent, such as greedy coordinate descent or cyclic coordinate descent. These algorithms differ in how a coordinate is chosen and updated. In this chapter we consider greedy coordinate descent. Greedy coordinate descent is an appealing choice, especially when one considers high-dimensional optimization problems. In this method a coordinate where the magnitude of gradient is largest is selected. This greedy choice ensures maximal change in the objective function under the constraint of updating a single coordinate per iteration. Here we briefly review the greedy coordinate descent algorithm and present convergence guarantees for it. These results are similar to those previously reported in Clarkson (2008). We assume that the

Algorithm 5 Greedy coordinate descent for minimizing convex function $J(\theta)$ over simplex Δ_1.

```
1: Initialization:
2: Choose $\theta^{(0)}$ arbitrarily.
3: $k \leftarrow 0$
4: repeat
5:     $k \leftarrow k+1$
6:     $I = \arg\max_{i \in \mathcal{I}} \langle -\nabla J(\theta^{(k-1)}), e_i \rangle$
7:     Set the learning rate $\eta_k = \frac{2}{k+1}$,
       or alternatively choose $\eta_k = \arg\min_{\eta \in [0,1]} J((1-\eta)\theta^{(k-1)} + \eta e_I)$
8:     $\theta^{(k)} \leftarrow (1-\eta_k)\theta^{(k-1)} + \eta_k e_I$
9: until convergence
```

goal is to minimize a convex function over a simplex:

$$\begin{aligned} \min_{\theta} \quad & J(\theta), \\ \text{s.t.} \quad & \theta \in \Delta_1. \end{aligned} \tag{5.1}$$

Algorithm 5 shows the GCD algorithm for solving optimization problem (5.1). This algorithm is also known as the Frank-Wolfe method (Frank and Wolfe, 1956) and the conditional gradient method. Here, e_i is the basis vector in which the i^{th} coordinate is 1, and the rest are zero. Theorem 5.1 shows that Algorithm 5 achieves an ε-optimal solution after $O(\frac{1}{\varepsilon})$ iterations. This result, however, is not new and similar bounds have been reported numerous times in the literature (e.g. in Zhang (2003); Clarkson (2008); Jaggi (2013)).

Theorem 5.1. *For each $k \geq 1$, the sequence $\theta^{(k)}$ obtained by Algorithm 5 satisfy*

$$J(\theta^{(k)}) - J(\theta^*) \leq \frac{2C_J}{k+2}, \tag{5.2}$$

where θ^* is the minimizer of function J over simplex Δ_1, J is assumed to be convex and differentiable, where for all $x, z \in \Delta_1$ and $\eta \in [0,1]$ it satisfies

$$J((1-\eta)x + \eta z) \leq J(x) + \eta \langle \nabla J(x), z - x \rangle + \frac{C_J}{2}\eta^2,$$

where C_J is a measure of curvature of J.

Note that, e.g., for optimization problem (2.12), C_J does not depend on the number of dimensions of θ. One consideration about the GCD algorithm is determining the step size (Step 7 in Algorithm 5). While for some objective functions, such as Euclidean distance, it is easy to find the best step size, it may be difficult or expensive to find the optimal step size in the general case. The proof of the theorem is given in Appendix A for the easier case of setting the learning rate to $\eta_k = \frac{2}{k+1}$. However it is obvious that finding the optimal learning rate through line search only results in a faster convergence of the algorithm.

Another consideration regarding Algorithm 5 is that obtaining the exact solution of the sub-problem in line 6 can be too expensive in some domains. Hence, it is desirable to study the behavior of the algorithm when one only finds an approximate maximizer of $\left\langle -\nabla J(\theta^{(k-1)}), e_i \right\rangle$. Jaggi (2013) studied the case where an ε-optimal solution is found in each iteration. In this case the error bound becomes:

$$J(\theta^{(k)}) - J(\theta^*) \leq \frac{2C_J}{k+2} + 2\varepsilon.$$

5.2 Greedy Coordinate Descent for One-Stage MKL

We aim to solve problem (2.9) of one-stage MKL using the greedy coordinate descent algorithm described in Section 5.1. Greedy coordinate descent is an appealing algorithm to solve MKL problems when one considers large sets of kernels as they update one coordinate (in this case, kernel coefficient) per iteration. Just like our approach to solve one-stage MKL with stochastic gradient descent, described in Section 4.2, we exploit the joint convexity of $J(w, \theta)$ and aim to minimize $J(\theta)$ defined in (4.7). All of the properties of $J(\theta)$ that we mentioned in Section 4.2 hold here too. Therefore, we do not repeat them here. For instance, the gradient of $J(\theta)$, which is required here, is given in (4.12).

Given the dual variables $\alpha^*(\theta^{(k-1)})$, and the form of gradient in (4.12), step 6 in Algorithm 5 results finding coordinate I that satisfies:

$$I = \underset{i \in \mathcal{I}}{\arg\max} \; \frac{1}{\rho_i^2} \left\langle \mathbf{K}_i, \alpha^*(\theta^{(k-1)}) \alpha^*(\theta^{(k-1)})^\top \right\rangle_F . \tag{5.3}$$

When the number of dimensions of θ is small or moderate, one can find the best coordinate by simply iterating over all coordinates. However, if the size of θ is large, one needs to seek other alternatives. Fortunately, in some cases, choosing the best kernel index can be transformed into an optimization problem that depends on the form of the underlying kernel family. For instance, if base kernels are Gaussians with bandwidth parameter σ, problem (5.3) becomes

$$\sigma^* = \underset{\sigma \in \Sigma}{\arg\max} \; \frac{1}{\rho_\sigma^2} \left\langle \mathbf{K}_\sigma, \alpha^*(\theta^{(k-1)}) \alpha^*(\theta^{(k-1)})^\top \right\rangle_F , \tag{5.4}$$

Algorithm 6 Greedy coordinate descent for one-stage MKL.

```
1: Initialization:
2: $\theta^{(0)} \leftarrow 0$
3: $k \leftarrow 0$
4: repeat
5:     $k \leftarrow k + 1$
6:     Compute $\alpha^*(\theta^{(k-1)})$ according to (4.9)
7:     $I = \arg\max_{i \in \mathcal{I}} \frac{1}{\rho_i^2} \left\langle \mathbf{K}_i, \alpha^*(\theta^{(k-1)})\alpha^*(\theta^{(k-1)})^\top \right\rangle_F$
8:     $\eta_k = \arg\min_{\eta \in [0,1]} J((1-\eta)\theta^{(k-1)} + \eta e_I)$
9:     $\theta^{(k)} = (1-\eta_k)\theta^{(k-1)} + \eta_k e_I$
10: until convergence
```

where $\rho_\sigma > 0$, $\sigma \in \Sigma$ are assumed to be given. The next step is to find the best step size for the new kernel. As we mentioned earlier, one can resort to the predefined learning rate $\eta_k = \frac{2}{k+3}$ for the general case. However, for many loss functions that are used in one-stage MKL, it is possible to compute the best learning rate through line search. Algorithm 6 shows the GCD algorithm for one-stage MKL.

The greedy coordinate descent approach for one-stage MKL has been previously proposed in Argyriou et al. (2005). In fact their algorithm is the same as Algorithm 6. Another similar work is presented in Gehler and Nowozin (2008). The only slight difference is that in Gehler and Nowozin (2008) once a kernel is selected, in addition to learning the coefficient of the last kernel, the coefficients of all previous kernels are updated too. This can be done by any of the existing finite MKL algorithms.

5.3 Greedy Coordinate Descent for Two-Stage MKL

In this section, we focus on applying greedy coordinate descent to two-stage MKL. In particular, we aim to solve the alignment maximization problem. Alignment maximization was discussed as a prominent approach to two-stake MKL in Section 2.2.2. In this section, we present a greedy coordinate descent algorithm for the alignment maximization problem. Our algorithm can be viewed as a steepest ascent approach to forward stagewise additive modeling (Hastie et al., 2001). As we mentioned in Section 2.2.2 alignment is not a convex objective function. However, one can solve an equivalent Euclidean distance minimization problem that gives an alignment maximizing solution. We, therefore, present a greedy coordinate descent algorithm in Section 5.3.2 to solve the Euclidean distance minimization problem. When we aim to solve a maximization problem, we will use the term coordinate ascent. We begin by presenting a steepest ascent algorithm for the alignment maximization problem.

5.3.1 Alignment maximization

Greedy coordinate ascent for alignment maximization can be viewed as a steepest ascent approach to forward stagewise additive modeling (FSAM). FSAM (Hastie et al., 2001) is an iterative method for optimizing an objective function by sequentially adding new basis functions without changing the parameters and coefficients of the previously added basis functions. Despite this subtle difference, this method still falls in the category of greedy coordinate descent algorithms.

Since centered alignment is used as the objective function in this section, and it is important to address centering of kernel matrices, it is preferable to derive and present the algorithm as updating the kernel combination rather than the kernel coefficient vector θ used throughout this thesis. We also assume that we aim to learn a combination of parameterized base kernels, i.e., greedy coordinate descent is applied in a space where coordinates are parameterized base kernels κ_σ.

Let $f(\kappa) = \hat{A}_c(\mathbf{K}(\kappa), \hat{\mathbf{K}}^*)$ denote the centered alignment function, where $\hat{A}_c, \mathbf{K}(\kappa)$, and $\hat{\mathbf{K}}^*$ were defined in Section 2.2.2. In the steepest ascent approach, in iteration k, the algorithm searches for the base kernel in $(\kappa_\sigma)_{\sigma\in\Sigma}$ defining the direction in which the growth rate of f is the largest, locally in a small neighborhood of the previous candidate $\kappa^{(k-1)}$:

$$\sigma_k^* = \operatorname*{arg\,max}_{\sigma\in\Sigma} \lim_{\varepsilon\to 0} \frac{f(\kappa^{(k-1)} + \varepsilon\,\kappa_\sigma) - f(\kappa^{(k-1)})}{\varepsilon}\,. \tag{5.5}$$

Once σ_k^* is found, the algorithm finds a coefficient $0 \le \eta_t \le \eta_{\max}$ [1] such that $f(\kappa^{(k-1)} + \eta_k \kappa_{\sigma_k^*})$ is maximized and the kernel combination is updated using

$$\kappa^{(k)} = \kappa^{(k-1)} + \eta_k \kappa_{\sigma_k^*}.$$

The process stops when the objective function f ceases to increase by an amount larger than a specified threshold $\delta > 0$, or when the number of iterations becomes larger then a predetermined limit T, whichever happens earlier.[2] Let us define

$$F_c(\mathbf{K}) \overset{\text{def}}{=} A_c(\mathbf{K}, \hat{\mathbf{K}}^*),$$

[1] The value of $\eta_{\max}$ acts as a regularizer. In the experiments, we use the arbitrary value $\eta_{\max} = 1$.

[2] We set $T = 50$ in experiments. We noticed that in practice the procedure always stops before the limit T on the number of iterations is reached.

and

$$F(\mathbf{K}) \stackrel{\text{def}}{=} \frac{\left\langle \mathbf{K}, \hat{\mathbf{K}}_c^* \right\rangle_F}{\|\mathbf{K}\|_F \|\hat{\mathbf{K}}_c^*\|_F},$$

so that $F_c(\mathbf{K}) = F(\mathbf{K}_c)$.

Proposition 5.2. *The value of σ_k^* can be obtained by*

$$\sigma_k^* = \arg\max_{\sigma \in \Sigma} \left\langle \mathbf{K}(\kappa_\sigma), F'(\,(\mathbf{K}(\kappa^{(k-1)}))_c) \right\rangle_F , \tag{5.6}$$

where for a kernel matrix $\mathbf{K}$,

$$F'(\mathbf{K}) = \frac{\hat{\mathbf{K}}_c^* - \|\mathbf{K}\|_F^{-2} \left\langle \mathbf{K}, \hat{\mathbf{K}}_c^* \right\rangle_F \mathbf{K}}{\|\mathbf{K}\|_F \|\hat{\mathbf{K}}_c^*\|_F} . \tag{5.7}$$

Proof. First, notice that the limit in (5.5) is a directional derivative, $D_{\kappa_\sigma} f(\kappa^{(k-1)})$. By the chain rule,

$$D_{\kappa_\sigma} f(\kappa^{(k-1)}) = \left\langle \mathbf{K}(\kappa_\sigma),\ F_c'(\mathbf{K}(\kappa^{(k-1)})) \right\rangle_F .$$

Some calculations give that

$$F'(\mathbf{K}) = \frac{\hat{\mathbf{K}}_c^* - \|\mathbf{K}\|_F^{-2} \left\langle \mathbf{K}, \hat{\mathbf{K}}_c^* \right\rangle_F \mathbf{K}}{\|\mathbf{K}\|_F \|\hat{\mathbf{K}}_c^*\|_F}$$

(which is the function defined in (5.7)). We claim that the following holds:

Lemma 5.3. $F_c'(\mathbf{K}) = C_n F'(\mathbf{K}_c) C_n$.

Proof. By the definition of derivatives, as $H \to 0$,

$$F(\mathbf{K} + H) - F(\mathbf{K}) = \left\langle F'(\mathbf{K}), H \right\rangle_F + o(\|H\|).$$

Also,

$$F_c(\mathbf{K} + H) - F_c(\mathbf{K}) = \left\langle F_c'(\mathbf{K}), H \right\rangle_F + o(\|H\|).$$

Now,

$$\begin{aligned} F_c(\mathbf{K} + H) - F_c(\mathbf{K}) &= F(C_n\mathbf{K}C_n + C_nHC_n) - F(C_n\mathbf{K}C_n) \\ &= \left\langle F'(\mathbf{K}_c), C_nHC_n \right\rangle_F + o(\|H\|) \\ &= \left\langle C_nF'(\mathbf{K}_c)C_n, H \right\rangle_F + o(\|H\|), \end{aligned}$$

where the last property follows from the cyclic property of trace. Therefore, by the uniqueness of derivative,

$$F_c'(\mathbf{K}) = C_n F'(\mathbf{K}_c) C_n.$$

□

Now, notice that $C_n F'(\mathbf{K}_c) C_n = F'(\mathbf{K}_c)$. Thus, we see that the value of σ_k^* can be obtained by

$$\sigma_k^* = \arg\max_{\sigma\in\Sigma} \left\langle \mathbf{K}(\kappa_\sigma), F'(\,(\mathbf{K}(\kappa^{(k-1)}))_c) \right\rangle_F ,$$

which was the statement to be proved. □

The crux of the proposition is that the directional derivative in (5.5) can be calculated and gives the expression maximized in (5.6).

In general, the optimization problem (5.6) is not convex and the cost of obtaining a (good approximate) solution is hard to predict. Evidence that, at least in some cases, the function to be optimized is not ill-behaved is presented in Section 6.2.3. In our experiments in Chapter 6, an approximate solution to (5.6) is found using numerical methods.[3] As a final remark to this issue, note that, as is usual in boosting, finding the global optimizer in (5.6) might not be necessary for achieving good statistical performance.

The other parameter, η_k, however, is easy to find, since the underlying optimization problem has a closed form solution:

Proposition 5.4. *The value of η_k is given by*

$$\eta_k = \arg\max_{\eta\in\{0,\eta^*,\eta_{\max}\}} f(\kappa^{(k-1)} + \eta\kappa_{\sigma_k^*}),$$

where $\eta^* = \max(0, (ad-bc)/(bd-ae))$ if $bd - ae \neq 0$ and $\eta^* = 0$ otherwise, $a = \left\langle \mathbf{K}, \hat{\mathbf{K}}_c^* \right\rangle_F$, $b = \left\langle \mathbf{K}', \hat{\mathbf{K}}_c^* \right\rangle_F$, $c = \langle \mathbf{K}, \mathbf{K}\rangle_F$, $d = \langle \mathbf{K}, \mathbf{K}'\rangle_F$, $e = \langle \mathbf{K}', \mathbf{K}'\rangle_F$ and $\mathbf{K} = (\mathbf{K}(\kappa^{(k-1)}))_c$, $\mathbf{K}' = (\mathbf{K}(\kappa_{\sigma_k^*}))_c$.

Proof. Let $g(\eta) = f(\kappa^{(k-1)} + \eta\kappa_{\sigma_k^*})$. Using the definition of f, we find that with some constant $\rho > 0$,

$$g(\eta) = \rho\,\frac{a + b\eta}{(c + 2d\eta + e\eta^2)^{1/2}}.$$

Notice that here the denominator is bounded away from zero (this follows from the form of the denominator of f). In particular, $e > 0$. Further,

$$\lim_{\eta\to\infty} g(\eta) = -\lim_{\eta\to-\infty} g(\eta) = \rho\frac{b}{\sqrt{e}}. \tag{5.8}$$

Taking the derivative of g we find that

$$g'(\eta) = \rho\,\frac{bc - ad + (bd - ae)\eta}{(c + 2d\eta + e\eta^2)^{3/2}}.$$

[3] In particular, we use the fmincon function of Matlab

Algorithm 7 Forward stagewise additive modeling for alignment maximization with parametrized set of kernels.

```
1: Inputs:
2: kernel initialization parameter $\varepsilon$,
3: number of iterations $T$,
4: tolerance $\delta$,
5: maximum stepsize $\eta_{\max} > 0$.
6: Initialization:
7: $\mathbf{K}^{(0)} \leftarrow \varepsilon I_n$.
8: for $k = 1$ to $T$ do
9:     $P \leftarrow F'(\mathbf{K}^{(k-1)})$
10:    $P \leftarrow C_n \, P \, C_n$
11:    $\sigma^* = \arg\max_{\sigma \in \Sigma} \langle P, \mathbf{K}_\sigma \rangle_F$
12:    $\mathbf{K}' \leftarrow C_n \, \mathbf{K}_{\sigma^*} \, C_n$
13:    $\eta^* = \arg\max_{0 \le \eta \le \eta_{\max}} F(\mathbf{K}^{(k-1)} + \eta \mathbf{K}')$
14:    $\mathbf{K}^{(k)} \leftarrow \mathbf{K}^{(k-1)} + \eta^* \mathbf{K}'$
15:    if $F(\mathbf{K}^k) \le F(\mathbf{K}^{(k-1)}) + \delta$ then terminate
16: end for
```

Therefore, g' has at most one root and g has at most one global extremum, from which the result follows by solving for the root of g' (if g' does not have a root, g is constant). $\square$

The pseudocode of the full algorithm is presented in Algorithm 7. The algorithm needs the data, the number of iterations (T) and a tolerance (δ) parameter, in addition to a parameter ε used in the initialization phase and $\eta_{\max} > 0$. The parameter ε, which should have a very small value, is used in the initialization step to avoid division by zero, and its value has little effect on the performance. Note that the cost of computing a kernel-matrix, or the inner product of two such matrices is $O(n^2)$. Therefore, the complexity of the algorithm is quadratic in the number of samples. The actual cost will be strongly influenced by the computational cost of (5.6). We include actual running times in the experiments, which give a rough indication of the computational limits of the procedure.

5.3.2 Euclidean distance minimization

Given the ideal kernel $\hat{\mathbf{K}}_c^*$ built from data, in this section we aim to minimize the squared "distance" between $\hat{\mathbf{K}}_c^*$ and a linear combination of centered base kernels. We aim to solve

$$\min_{\theta \ge 0} \qquad J(\theta) = \frac{1}{2n^2} \left\| \beta \sum_{i \in \mathcal{I}} \theta_i \mathbf{K}_i - \hat{\mathbf{K}}_c^* \right\|_F^2 , \tag{5.9}$$

where $\beta > 0$ is a scaling factor. For simplicity of notation we assume that the base kernels are centered and we drop the subscript $_c$ from kernel matrices. We showed in Section 2.2.2 that the value of β has no effect on finding the alignment maximizing solution. However, since Algorithm 5 finds the optimal solution subject to the unit simplex constraint, choosing an appropriate value for β ensures that the optimal solution lies inside the unit simplex. It can be shown that choosing

$$\beta \geq \max_{i \in \mathcal{I}} \frac{\left\langle \mathbf{K}_i, \hat{\mathbf{K}}_c^* \right\rangle_F}{\left\langle \mathbf{K}_i, \mathbf{K}_j \right\rangle_F}, \quad \forall j \in \mathcal{I},$$

ensures that $\theta^* \in \Delta_1$. The details of the derivation and specific examples are given in Appendix B.

It is also easy to show that the curvature parameter C_J of the Euclidean distance loss function does not depend on the number of dimensions of θ. Hence, the bound given in Theorem 5.1 guarantees the convergence of Algorithm 5 when applied to two-stage MKL with very large kernel sets. See Appendix B for details.

The Euclidean distance minimization problem (5.9) is convex and can be solved by greedy coordinate descent. Let $\mathbf{K}^{(k-1)} = \beta \sum_{i \in \mathcal{I}} \theta_i^{(k-1)} \mathbf{K}_i$ be the linear combination of kernels selected up to iteration $k-1$. In iteration k, greedy coordinate descent chooses a coordinate $I^{(k)} \in \mathcal{I}$ such that

$$\begin{aligned} I^{(k)} &= \arg\min_{I \in \mathcal{I}} \lim_{\varepsilon \to 0} \frac{J(\theta^{(k-1)} + \varepsilon e_I) - J(\theta^{(k-1)})}{\varepsilon} \\ &= \arg\max_{I \in \mathcal{I}} \left\langle \mathbf{K}_I, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \right\rangle_F . \end{aligned}$$

Note that if

$$\left\langle \mathbf{K}_I, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \right\rangle_F \leq 0,$$

the algorithm must stop since the objective function cannot be further minimized. Once a coordinate $I^{(k)}$ is chosen, the algorithm computes the best step size through line search. Due to the nice form of problem (5.9) the optimization problem to find the optimal step size has a closed form solution. Note that

$$\begin{aligned} \eta^* &= \arg\min_{\eta} J(\theta^{(k-1)} + \eta e_{I^{(k)}}) \\ &= \frac{\left\langle \mathbf{K}_{I^{(k)}}, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \right\rangle_F}{\beta \left\langle \mathbf{K}_{I^{(k)}}, \mathbf{K}_{I^{(k)}} \right\rangle_F}. \end{aligned}$$

Furthermore, note that $\eta^* > 0$ since $\langle \mathbf{K}_I, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \rangle_F > 0$. Hence, the best step size using line search is obtained by

$$\eta_k = \min(\eta^*, 1) .$$

Algorithm 8 Greedy coordinate descent for two-stage MKL using Euclidean distance minimization. For the definition of $\hat{\mathbf{K}}_c^*$, β, and C_n see the text.

```
1: **Initialization:**
2: $\theta^{(0)} \leftarrow 0$
3: $k \leftarrow 0$
4: **repeat**
5:     $k \leftarrow k+1$
6:     $\tilde{\mathbf{K}} \leftarrow \beta \sum_{i \in \mathcal{I}} \theta_i^{(k-1)} \mathbf{K}_i$
7:     $\mathbf{K}^{(k-1)} \leftarrow C_n \tilde{\mathbf{K}} C_n$ // centering
8:     $I = \arg\max_{i \in \mathcal{I}} \langle \mathbf{K}_i, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \rangle_F$
9:     **if** $\langle \mathbf{K}_I, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \rangle_F \leq 0$ **then** terminate
10:    $\eta_k \leftarrow \min(1, \beta^{-1} \langle \mathbf{K}_I, \hat{\mathbf{K}}_c^* - \mathbf{K}^{(k-1)} \rangle_F / \langle \mathbf{K}_I, \mathbf{K}_I \rangle_F)$
11:    $\theta^{(k)} \leftarrow (1-\eta_k)\theta^{(k-1)} + \eta_k e_I$
12: **until** convergence
```

Finally, the kernel weight vector is updated by

$$\theta^{(k)} = (1-\eta_k)\theta^{(k-1)} + \eta_k e_{I^{(k)}}.$$

Algorithm 8 shows the greedy coordinate descent algorithm for solving problem (5.9).

5.4 Summary

In this chapter, we discussed greedy coordinate descent for multiple kernel learning. Greedy coordinate descent is particularly useful when one considers parameterized kernel families with infinitely many kernels. One prominent and important family of such kernels is Gaussian kernels. We presented greedy coordinate descent algorithms for one-stage MKL as well as two-stage MKL based on alignment maximization and Euclidean distance minimization. We will evaluate the performance of greedy coordinate descent, in one-stage and two-stage MKL in Chapter 6.

Chapter 6

Experimental Results

In this chapter, we present the experimental results of stochastic gradient descent and greedy coordinate descent for multiple kernel learning. In particular, we are interested in learning polynomial and Gaussian kernels. These are important family of kernels that have been used extensively in kernel methods. For our experiments, polynomial kernels are well suited for the stochastic gradient descent learning scheme. We presented an efficient learning algorithm for polynomial kernels in Section 4.2.1. Parameterized Gaussian kernels, on the other hand, are well suited for the greedy coordinate descent learning scheme. Choosing the best coordinate simply translates into solving an optimization problem over the kernel parameters, which in case of Gaussian kernels, is finding the best kernel bandwidth.

This chapter is organized as follows. In Section 6.1, we present the experimental results of stochastic gradient descent for one-stage multiple kernel learning. In this section, we mainly focus on learning polynomial kernels, with the exception of Section 6.1.5, in which, we briefly explore the application of stochastic gradient descent to learning Gaussian kernels. Then, in Section 6.2, we present the experimental results of greedy coordinate descent for learning Gaussian kernels. In this section we consider one-stage and two-stage kernel learning settings. Finally, in Section 6.3, we compare stochastic gradient descent and greedy coordinate descent methods for multiple kernel learning. In this chapter we compare our algorithms against various state-of-the-art MKL algorithms. Table 6.1 shows the list of algorithms we examine in different experiments in this chapter along with their acronyms.

Table 6.1: Various MKL algorithms considered in experiments.

Acronym	**Description**
SGD–1s	Stochastic gradient descent for one-stage MKL, (cf. Section 4.2)
GCD–1s	Greedy coordinate descent for one-stage MKL, (cf. Section 5.2)
GCD–2s	Greedy coordinate descent for two-stage MKL, (cf. Section 5.3)
LpMKL	p-norm MKL (Kloft et al., 2011)
HMKL	Hierarchical MKL (Bach, 2008)
NLMKL	Non-linear MKL (Cortes et al., 2009b)
CKL	Continuously-parameterized kernel learning (Argyriou et al., 2005)
IKL	Infinite kernel learning (Gehler and Nowozin, 2008)
AMKL	Alignment-based two-stage MKL (Cortes et al., 2010)
Uniform	Linear combination of base kernels with uniform weights

6.1 Stochastic Gradient Descent for One-Stage MKL

In this section, we explore the effectiveness of stochastic gradient descent in one-stage MKL, proposed in Section 4.1. We perform several experiments to this end. First, in Section 6.1.1, we compare several methods for sampling, which can be used as part of the stochastic gradient descent algorithm. We design an experiment in which we show the effectiveness of gradient-based sampling over other alternatives. We then repeat a similar experiment, in a multiple kernel learning problem in Section 6.1.2. After that, we focus on comparing algorithms in terms of prediction error, and training time, using synthetic data, in Section 6.1.3, and using real data, in Section 6.1.4. In the above experiments we aim to learn polynomial kernels (cf. Section 4.2.1). At the end of this section, we focus on learning Gaussian kernels. We demonstrate how our algorithm can deal with continuously parametrized, uncountable kernel sets, to learn a linear combination of Gaussian kernels. We consider learning a combination of Gaussian kernels when they are parameterized by a single parameter chosen from an interval, as well as when they are parameterized by a vector of parameters chosen from a rectangle in a multi-dimensional space. We follow the algorithms proposed in Section 4.2.2 for learning Gaussian kernels.

Unless stated otherwise, throughout this section, we consider the problem of multiple kernel learning in regression with the squared loss:

$$\ell(w) = \frac{1}{2}\sum_{t=1}^{n}(f_w(x_t) - y_t)^2,$$

where $\{(x_t, y_t) \in \mathbb{R}^r \times \mathbb{R}\}_{t=1}^{n}$ is the training dataset that consists of input-output

Algorithm 9 Nesterov's RCDM algorithm (Nesterov, 2010)

```
1: Inputs:
2: Lipschitz constants of gradient of objective function $f$: $L_i, i \in \{1, \ldots, r\}$,
3: parameter $\alpha$.
4: Initialization:
5: Choose $x^{(0)}$ arbitrarily
6: $k \leftarrow 0$
7: repeat
8:     $k \leftarrow k + 1$
9:     Sample $I_k$ from $P(\cdot)$, where $P(i) = L_i^\alpha / \sum_{j=1}^r L_j^\alpha$
10:    $x^{(k)} \leftarrow x^{(k-1)} - \frac{1}{L_{I_k}} \left\langle \nabla f(x^{(k-1)}), e_{I_k} \right\rangle e_{I_k}$
11: until convergence
```

pairs. Prediction accuracy values are reported as mean squared error over test sets. A constant feature is added to act as offset, and the inputs and output are normalized to have zero mean and unit variance. Each experiment was performed with 10 runs in which we randomly choose training, validation, and test sets. The results are averaged over these runs.

6.1.1 Effect of sampling method

The sampling procedure for stochastic gradient methods, proposed in Section 4.1, is not solely applicable to MKL. This sampling procedure uses a distribution that is proportional to the magnitude of gradient. In this experiment, we further investigate the virtues of this method of sampling. To do so, we compare Nesterov's RCDM algorithm (Nesterov, 2010, Alg. (2.6)) with $\alpha = 1$, Nesterov's RACDM algorithm(Nesterov, 2010, Alg. (6.1)), which dynamically adjusts the Lipschitz constants of gradient[1], and our method, which stochastically chooses coordinates based on the magnitude of gradient coordinates. We denote the latter Grad.Samp. here. The assumption of Nesterov's RCDM is that the gradient of the objective function be Lipschitz continuous. Let L_i denote the Lipschitz constant of gradient along the i^{th} coordinate. Let f be the objective function. The RCDM algorithm is shown in Algorithm 9. In this experiment, we aim to minimize the objective function given by

$$f(x) = \frac{1}{2} \sum_{i=1}^{10^4} (x_i - 0.9^i)^2 \,. \tag{6.1}$$

[1]This method is useful when Lipschitz constants of gradient are unknown or it is hard to estimate an upper bound.

There are two differences between Nesterov's RCDM and our algorithm:

1. The learning rate of Nesterov's method is $1/L_i$ (See Algorithms (2.6) and (6.1) in Nesterov (2010)). On the other hand, our method uses a constant learning rate of $1/\sqrt{T}$, where T is a parameter that specifies the number of iterations.[2] This seems to be in favor of Nesterov's method. For example, in minimizing the objective function (6.1), once a coordinate is selected randomly, Nesterov's method is able to learn the optimal value of that coordinate in one update. On the other hand, the learning rate used by our method provides a gradual convergence towards the optimal value.

2. Nesterov's method uses the Lipschitz constants of *gradient* to define the sampling distribution. In the above example these constants are equal for all coordinates ($L_i = 1$, $i \in \{1, \ldots, 10^4\}$), which results in selecting coordinates from a uniform distribution. On the other hand, our method uses a sampling distribution which is proportional to the *magnitude* of gradient coordinates. This, in examples similar to (6.1), gives higher probabilities to more important coordinates.

We run all methods for 2×10^4 iterations. Figure 6.1 shows convergence speed of these algorithms. The vertical axis in these plots indicate the value of objective function divided by its initial value (when all variables are set to zero). Note the speed-up achieved due to the new sampling method used in Grad.Samp. compared to RCDM and RACDM.

6.1.2 Polynomial kernels – convergence test

In this experiment, we examine the speed of convergence of the SGD–1s method and compare it against one of the fastest standard multiple kernel learning algorithms, that is, the LpMKL algorithm of Kloft et al. (2011) with $p = 2$,[3] and the uniform coordinate descent algorithm that updates one coordinate, which is chosen uniformly at random, per iteration (Nesterov, 2010, 2012; Shalev-Shwartz and Tewari, 2011; Richtárik and Takáĉ, 2011). We denote the latter by UCD in this experiment. We aim to learn polynomial kernels of up to degree 3 with all algorithms. SGD–1s

[2]This learning rate was chosen according to the theoretical bound given for the algorithm in Section 4.1.

[3]Note that $p = 2$ in Kloft et al. (2011) notation corresponds to $p = 4/3$ or $\nu = 2$ in our notation, which gives the same objective function that we minimize with Algorithm 2.

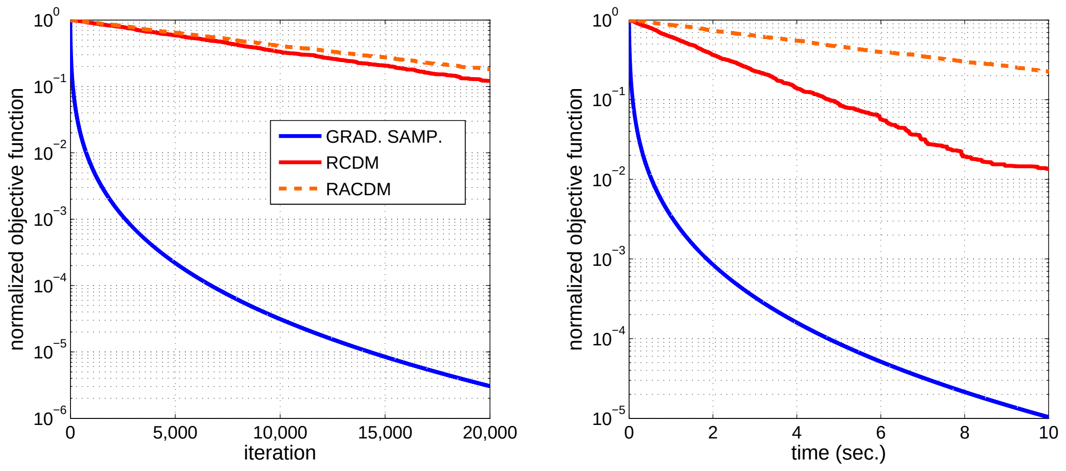

Figure 6.1: Convergence speed of algorithms.

uses Algorithm 3 for sampling with $D = 3$. The set of provided base kernels is the linear kernels built from input variables, that is, $\kappa_{(i)}(x, x') = x_{(i)}x'_{(i)}$, where $x_{(i)}$ denotes the i^{th} input variable. For the other two algorithms the kernel set consists of kernels from monomial terms for $D \in \{0, 1, 2, 3\}$ built from r base kernels, where r is the number of input variables. The total number of distinct base kernels (monomials) is $\binom{r+D}{D}$. In this experiment, for all algorithms, we use ridge regression with its regularization parameter set to 10^{-5}. Experiments with other values of the regularization parameter achieved similar results.

We compare these methods in four datasets from the UCI machine learning repository (Frank and Asuncion, 2010) and the Delve datasets[4]. The specifications of these datasets are given in Table 6.2. We run all algorithms for a fixed amount of time and measure the value of the objective function (2.7), that is, the sum of the empirical loss and the regularization term. Figure 6.2 shows the performance of these algorithms. The results show that SGD–1s consistently outperforms the other algorithms in convergence speed. Note that our stochastic method updates one kernel coefficient per iteration, while LpMKL updates $\binom{r+D}{D}$ kernel coefficients per iteration. The difference between the two methods is analogous to the difference between stochastic gradient vs. full gradient updates. While UCD also updates one kernel coefficient per iteration its naive method of sampling coordinates results in a slower overall convergence compared to our algorithm. In the next section, we compare our algorithm against several representative methods from the MKL

[4]See, `www.cs.toronto.edu/~delve/data/datasets.html`.

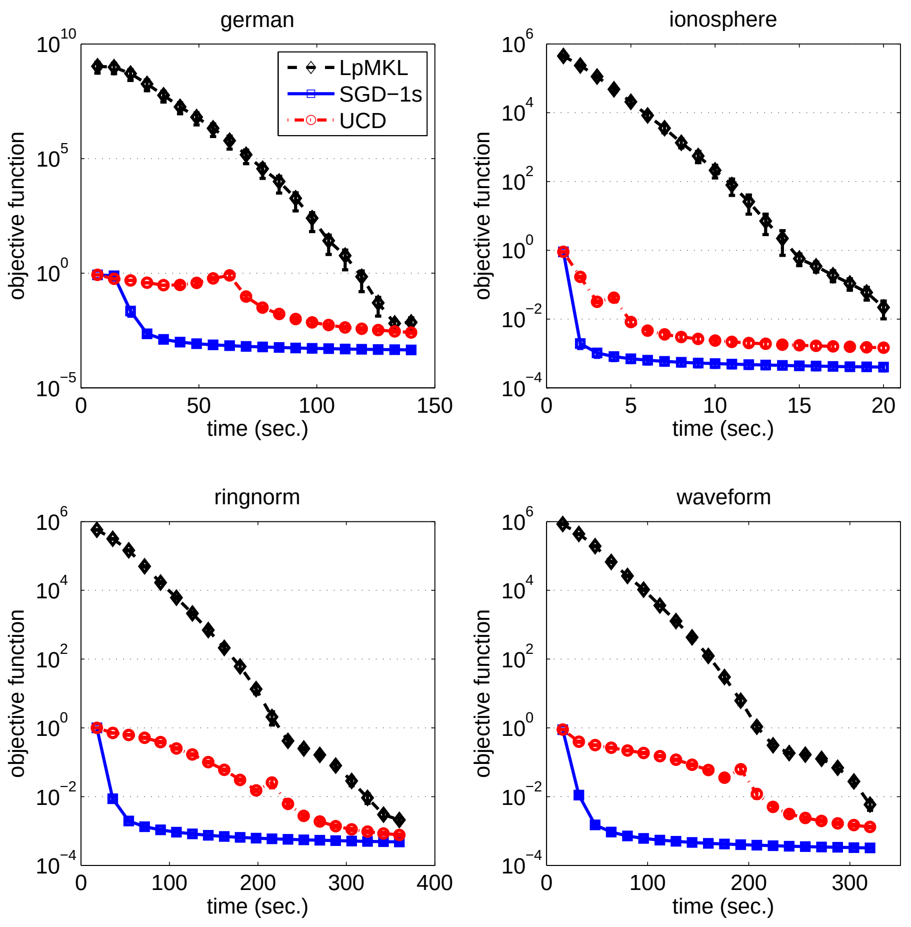

Figure 6.2: Convergence comparison of our method and other algorithms.

literature.

6.1.3 Polynomial kernels – synthetic data

In this experiment, we examine the effect of the size of the kernel space on prediction accuracy and training time of MKL algorithms. We generated data for a regression problem. Let r denote the number of dimensions of the input space. The inputs are chosen uniformly at random from $[-1, 1]^r$. The output of each instance is the uniform combination of 10 monomial terms of degree 3 or less built from the first 5 variables. These terms are chosen uniformly at random among all possible terms. The outputs are noise-free. We generated data for $r \in \{5, 10, 20, \ldots, 100\}$, with 500 training and 1000 test points. The regularization parameter of ridge regression was tuned from $\{10^{-8}, \ldots, 10^{2}\}$ using a separate validation set with 1000 data points.

We compare our method (SGD–1S) against LPMKL ($p = 2$), the non-linear kernel learning method of Cortes et al. (2009b) (NLMKL), the hierarchical kernel learning algorithm of Bach (2008) (HMKL). Recall that HMKL and NLMKL are specifically designed to learn polynomial kernels. For comparison purposes, we also compare these methods against a uniform combination of all non-linear kernels of degree 3 or less (UNIFORM). The set of base kernels consists of r linear kernels built from the input variables. Recall that the method of Cortes et al. (2009b) only considers kernels of the form $\kappa_\theta = (\sum_{i=1}^{r} \theta_i \kappa_i)^D$, where D is a predetermined integer that specifies the degree of polynomial kernel. Note that adding a constant feature results in adding polynomial kernels of degree less than D to the combination too. We provide all possible polynomial kernels of degree 0 to D to the kernel learning method of Kloft et al. (2011). For our method and the method of Bach (2008) we set the maximum kernel degree to $D = 3$. Recall that our method (and HMKL) select kernels from a set with $(r+1)^D$ kernels.

The results are shown in Figure 6.3, the mean squared errors are on the left plot, and the training times are on the right plot. In the training-time plot the numbers inside brackets indicate the total number of distinct kernels for each value of r. This is the number of kernels fed to the LPMKL algorithm. Since this method deals with a large number of base kernels, it was possible to precompute and keep the kernels in memory (8GB) for $r \leq 25$. Therefore, we ran this algorithm for $r \leq 25$. For higher number of variables, we could use on-the-fly implementation of this algorithm, however that further increases the training time. Note that the computation cost of this method is cubic ($D = 3$) in the size of the input space (or linearly in the number of kernels). While the standard MKL algorithms, such as LPMKL, cannot handle very large kernel spaces, in terms of time and space complexity, the other three algorithms can efficiently learn kernel combinations. Their predictive accuracies, however, are quite different. While NLMKL performs well for small number of variables, its performance starts to degrade as r increases. This is due to the restricted family of kernels that this method considers. The HMKL method, which is well-suited to learn sparse combination of polynomial kernels, performs better than NLMKL for higher input dimensions. Among all methods, however, our method performs best in predictive accuracy while its computational cost is only slightly worse than the other two main competitors.

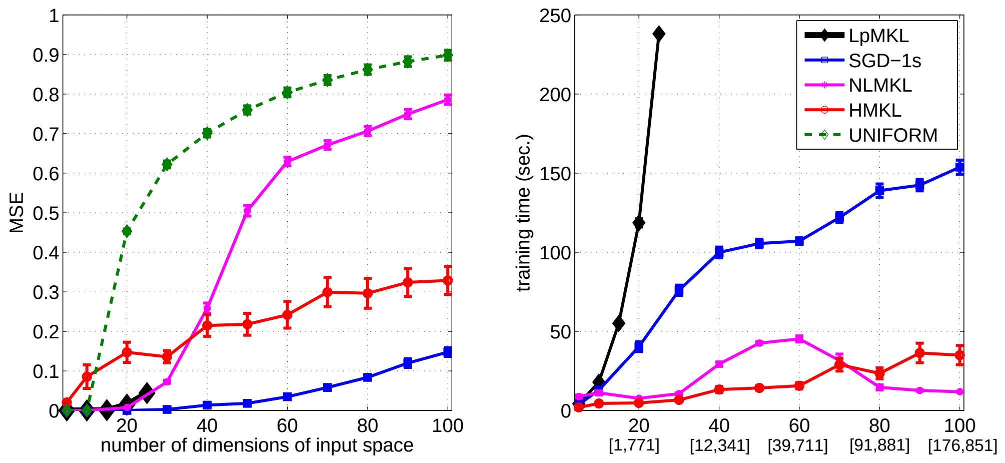

Figure 6.3: Comparison of kernel learning methods in terms of test error (left) and training time (right) on the synthetic dataset.

6.1.4 Polynomial kernels – real data

In this experiment, we consider again the polynomial kernels that are built from linear kernels corresponding to the input variables. In this case we chose several datasets from the UCI machine learning repository and Delve datasets. We selected datasets in which the number of dimensions of the input space is 20 or more. We aim to compare polynomial kernels against linear kernels. We consider polynomial kernels of degree 2 and 3 in this experiment. The following list shows the methods along with the parameters used in this experiment:

- SGD–1s, with $D \in \{2, 3\}$.
- HMKL, with $D \in \{2, 3\}$.
- NLMKL, with $D \in \{2, 3\}$.
- LpMKL, with $p \in \{1, 2\}$. Base kernels are linear kernels.
- Uniform, uniform combination of all kernels up to degree D, i.e. $\kappa = (\sum_{i=1}^{r} \kappa_i)^D$, for $D \in \{1, 2, 3\}$.

The datasets include german, ringnorm, waveform, ionosphere, sonar, and splice. Figure 6.4 shows the results. The specifications of these datasets are given in Table 6.2. The regularization parameter of ridge regression is tuned from the set $\{10^{-4}, \ldots, 10^3\}$ for all methods using a validation set.

Table 6.2: Specifications of datasets used in the experiments.

Dataset	**# of variables**	**Train. size**	**Valid. size**	**Test size**
banana	2	500	1000	2000
breast cancer	9	105	27	131
diabetes	8	307	77	384
german	20	350	150	500
heart	13	108	27	135
image seg.	18	500	500	1000
ionosphere	34	140	36	175
ringnorm	20	500	1000	2000
sonar	60	83	21	104
splice	60	500	1000	1491
thyroid	5	86	22	107
waveform	21	500	1000	2000

Overall, we observe that methods that consider non-linear variable interactions (SGD–1s, HMKL, and NLMKL) perform better than linear methods (LpMKL). Among non-linear methods, NLMKL performs worse than the other two. We believe that this is due to the restricted kernel space considered by this method. The performance of SGD–1s and HMKL is similar overall.

We observe that our method overfits when it considers kernels of degree 3 in this experiment. However, one can easily address this issue by assigning higher ρ values to higher-degree kernels so that the algorithm selects lower-degree kernels more often. For this purpose, we ran this experiment for $D = 3$ with a modified set of ρ values, where we use $\rho_i^2 = 1$ for kernels of degree ≤ 2 and $\rho_i^2 = 4$ for kernels of degree 3. With the new ρ coefficients the algorithm was able to reduce overfitting. See SGD–1s ($D = 3$, prior) error values in Figure 6.4.

6.1.5 Gaussian kernels – real data

In this experiment we aim to learn a combination of Gaussian kernels with single- and multi-dimensional kernel parameters. We compare several algorithms, suitable for learning kernels from continuous sets, i.e. CKL and IKL, along with our algorithm, SGD–1s. These methods are run with single-dimensional kernel parameter search (1D), and multi-dimensional kernel parameter search (nD). The method of Argyriou et al. (2005) has been improved in Argyriou et al. (2006) by applying DC-programming for parameter search. The DC-programming version, however, is

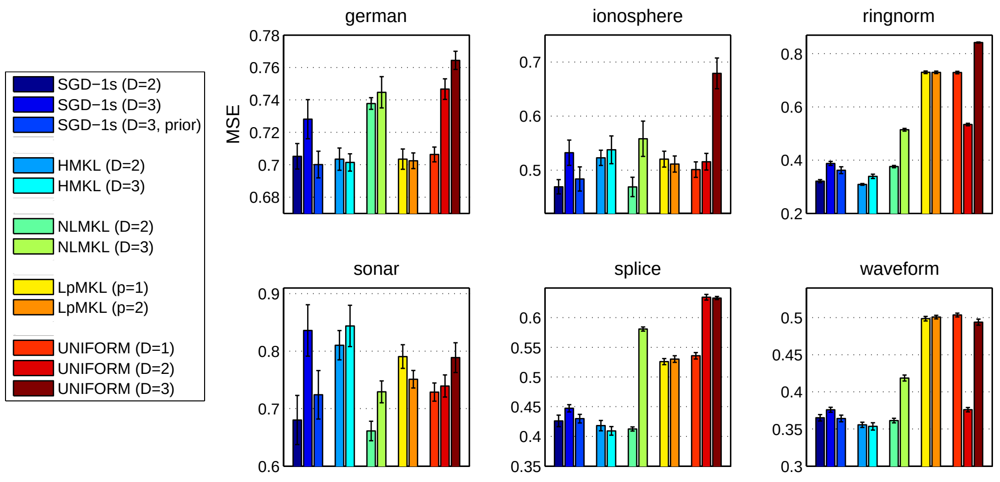

Figure 6.4: Prediction error of different methods in the real data experiment

not scalable to multi-dimensional search. In this experiment we implemented the original version proposed in Argyriou et al. (2005) in order to be able to run the method for multi-dimensional parameter search. We restrict the parameter search to interval $\Sigma = [10^{-4}, 10^{4}]$. We use Matlab's fmincon function to solve the parameter search sub-problem for these kernel learning methods. We also run LpMKL ($p = 2$) with 50 kernels selected by discretization of the above interval in a geometric fashion, where the bandwidth parameter of the i^{th} kernel is equal to $10^{-4} \cdot (1.6)^{i-1}$. Finally, we evaluate the performance of a uniform combination of these 50 kernels (Uniform).

In this experiment, we consider 11 datasets. The specifications of the datasets are given in Table 6.2. The regularization parameter of ridge regression is tuned from the set $\{10^{-4}, \ldots, 10^{3}\}$ using 5-fold cross validation. The stopping criterion for all of the methods is that the change in the value of the objective function in consecutive iterations is less than 1% of its current value. We found that no further improvement is made by using finer stopping thresholds. The results are shown in Figure 6.5. The results suggest that in most datasets it is better to search for one-dimensional kernel parameter. However, there are cases, such as image and thyroid, in which multi-dimensional kernel parameter search achieves better performance. In general the performance of SGD–1s (in particular, the (1D) version) is comparable to that of other infinite kernel learning methods, as well as that of LpMKL. The median rank of compared algorithms over all datasets, shown in Table 6.3, also

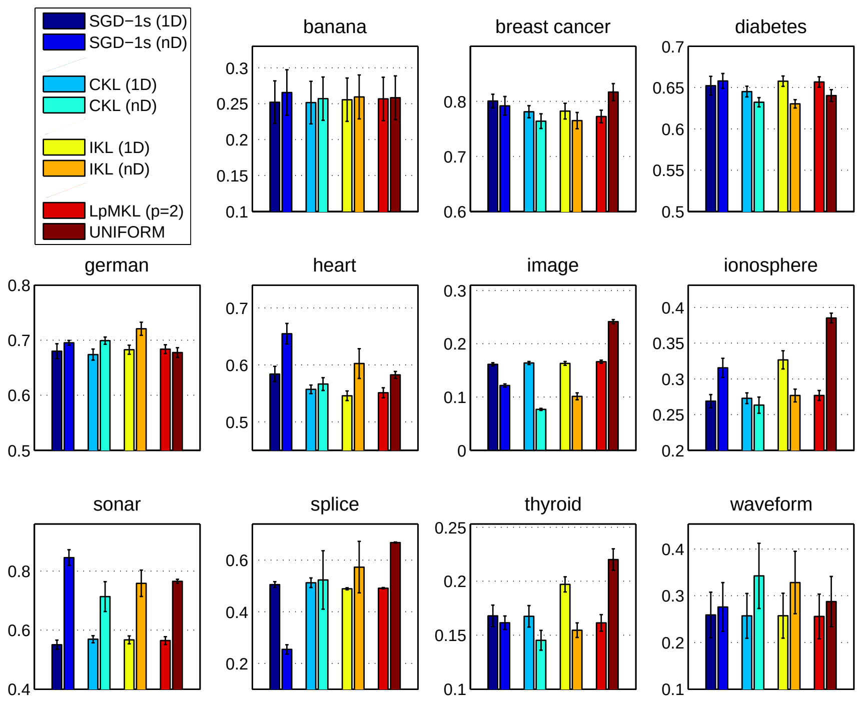

Figure 6.5: Prediction error of different methods in the Gaussian kernel experiment

confirms this.

6.2 Greedy Coordinate Descent for MKL

In this section, we compare our greedy coordinate descent MKL against several kernel learning methods on synthetic and real data. In particular, we use Algorithm 6 for one-stage MKL and Algorithm 8 for two-stage MKL. Recall that Algorithm 6 has been previously proposed in Argyriou et al. (2005), which we denoted in previous experiments by CKL. This makes GCD–1s and CKL algorithms identical.

Table 6.3: Median rank of algorithms in the Gaussian kernel experiment.

SGD–1s (1D)	SGD–1s (nD)	CKL (1D)	CKL (nD)	IKL (1D)	IKL (nD)	LpMKL	Uniform
3	6	3	3.5	3.5	5	3	6.5

We, therefore, do not run a separate GCD–1S algorithm. Our method for two-stage MKL is denoted by GCD–2S in this section. In all of the experiments, we run CKL and GCD–2S until either (i) the number of iterations reaches 50, or (ii) the change in the value of objective function is less than 10^{-3}, whichever comes first.

As we mentioned, greedy coordinate descent is an appealing approach for combining kernels when base kernels belong to a continuously-parameterized set. We, therefore, experiment with parameterized Gaussian kernels. In Section 6.2.1, we use synthetic data to illustrate the potential advantage of methods that work with a continuously parameterized set of kernels. We also illustrate in a toy example that multi-dimensional kernel parameter search can be advantageous in certain problems. We also investigate scalability of these methods in this section. These are followed by the evaluation of various MKL methods on real datasets in Section 6.2.2.

6.2.1 Gaussian kernels – synthetic data

In this section, we design a dataset to illustrate that when it comes to continuously parameterized kernels, in some cases, multi-dimensional parameter search is crucial for good performance. The dataset is designed as follows: The instances for the positive (negative) class are generated from a $r = 50$-dimensional Gaussian distribution with covariance matrix $\mathbf{C} = \mathbf{I}_{r\times r}$ and mean $\mu_1 = \rho\frac{\beta}{\|\beta\|}$ (respectively, $\mu_2 = -\mu_1$ for the negative class). Here $\rho = 1.75$. The vector $\beta \in [0,1]^r$ determines the relevance of each feature in the classification task, e.g. $\beta_i = 0$ implies that the distributions of the two classes have zero means in the i^{th} feature, which renders this feature irrelevant. The value of each component of vector β is calculated as $\beta_i = (i/r)^\gamma$, where γ is a constant that determines the relative importance of the elements of β. We generate seven datasets with $\gamma \in \{0, 1, 2, 5, 10, 20, 40\}$. For each value of γ, the training set consists of 50 data points (the prior distribution for the two classes is uniform). The test error values are measured on a test set with 2000 instances. We repeated each experiment 10 times and report the average misclassification error and alignment measured over the test set along with the training time.

In this experiment, we use greedy coordinate descent with one-dimensional and multi-dimensional Gaussian kernels. The form of these kernels are shown in (4.19) and (4.20) respectively. When the size of training set is small, as in this experiment, multi-dimensional parameter search may overfit. To address overfitting we modify the algorithm to shrink the values of the bandwidth parameters to their common average value by modifying (5.6):

$$\sigma_k^* = \arg\min_{\sigma \in \Sigma} -\left\langle \mathbf{K}(\kappa_\sigma), F'(\,(\mathbf{K}(\kappa^{(k-1)}))_c) \right\rangle_F + \lambda \|\sigma - \bar{\sigma}\|_2^2, \tag{6.2}$$

where, $\bar{\sigma} = \frac{1}{r}\sum_{i=1}^{r}\sigma_i$ and λ is a regularization parameter. For the sake of completeness we also include the unregularized version of GCD–2S (nD) in which $\lambda = 0$. It is labeled GCD–2S (nD,NoReg) in plots. The CKL algorithm has also been run with single- and multi-dimensional parameter search procedures, denoted by CKL (1D) and CKL (nD) respectively. We also include results obtained by finite MKL methods. For these methods, we generate 50 Gaussian kernels with one-dimensional bandwidths $\sigma \in mg^{\{0,\ldots,49\}}$, where $m = 10^{-3}$, and $g \approx 1.33$. Therefore, the bandwidth range constitutes a geometric sequence from 10^{-3} to 10^{3}.

The one-dimensional versions, i.e. CKL (1D) and GCD–2S (1D), employ Matlab's fmincon function with multiple restarts from the set $10^{\{-3,\ldots,5\}}$, to choose the kernel parameters. The multi-dimensional versions, CKL (nD) and GCD–2S (nD), use fmincon only once, since in this particular example the search method runs on a 50-dimensional search space, which is an expensive operation. The starting point of multi-dimensional search methods is a vector of equal elements where this element is the weighted average of the kernel parameters found by the corresponding one-dimensional search method, weighted by the coefficient of the corresponding kernels. Since this is a classification problem we train the classifier with soft margin SVM. The regularization parameter of SVM is tuned from the set $10^{\{-5,-4.5,\ldots,4.5,5\}}$ using an independent validation set with 1000 instances. We also tuned the value of the regularization parameter in problem (6.2) from $10^{\{-5,\ldots,14\}}$ using the same validation set (the best value of λ is the one that achieves the highest value of alignment on validation set). We decided to use a large validation set, following essentially the practice of Kloft et al. (2011, Section 6.1), to make sure that in the experiments reasonably good regularization parameters are used, i.e., to factor out the choice of the regularization parameters. This might bias our results towards GCD–2S (nD), compared to GCD–2S (1D), though similar results were achieved with a smaller validation set of size 200. As a final detail note that LPMKL ($p = 1$), LPMKL ($p = 2$), CKL (1D) and CKL (nD) also use the validation set for choosing the value of their regularization parameter, and together with the regularizer, the weights also. Hence, their results might also be positively biased (though we do not think this is significant, in this case).

In all experiments, training time corresponds to a single run of learning kernel weights and learning a classifier, i.e., the extra time required to tune SVM regularization parameter is not included. However, the time required to tune λ in (6.2) is considered in training time computation.

Figure 6.6 shows the results. Recall that the larger the value of γ, the larger is the number of nearly irrelevant features. Since methods that search only a one-dimensional space cannot differentiate between relevant and irrelevant features, their misclassification error increases with γ. On the other hand, multi-dimensional search methods are able to cope with this situation and even improve the performance. We observe that without regularization, however, for small values of γ, GCD–2s (nD,NoReg) and CKL (nD) drastically overfit. We also show the training time of the methods. Note that CKL (1D) is slower than GCD–2s (1D). [5] The same trend can be observed in their multi-dimensional counterparts, i.e. GCD–2s (nD,NoReg) and CKL (nD). However the training time of GCD–2s (nD) is comparable (in this experiment) to that of CKL (nD) since GCD–2s (nD) runs cross-validation to tune λ too. Although the large training time of multi-dimensional search methods might be prohibitive, for some problems, these methods might be the only option if good performance is crucial.

The (centered) alignment values for the learned kernels (on the test data) as a function of the relevance parameter γ are shown in Figure 6.7. It can be readily seen that multi-dimensional methods achieve higher alignment values when the number of irrelevant features is large. Note also that the discretization is fine enough so that the alignment maximizing finite kernel learning method AMKL can achieve the same alignment as GCD–2s (1D).

Scalability Test

Next, we compare scalability of these methods with respect to the number of training examples and the number of kernels. We choose the previous synthetic dataset with $\gamma = 40$ and increase the number of training examples and the number of kernels. As with the previous experiment we report misclassification error and training time. Figure 6.8 shows the results.

In the first experiment, we examine the methods when they are provided with

[5] Obviously, the training times are implementation-dependent and the comparisons should be taken with a grain of salt.

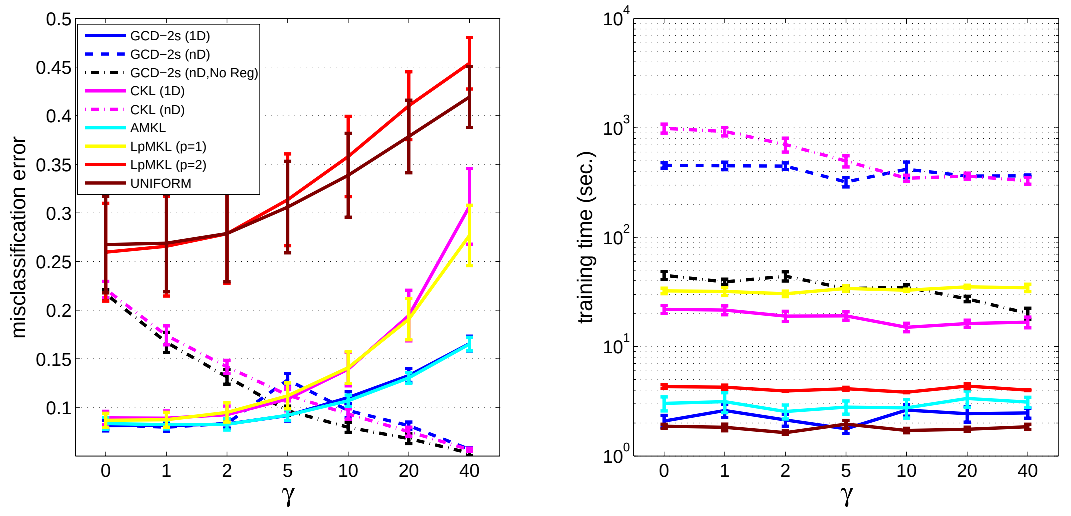

Figure 6.6: Misclassification error and training time of various methods in a 50-dimensional synthetic problem as a function of the relevance parameter γ. Note that the number of irrelevant features increases with γ. For details of the experiments, see the text.

different number of training examples. While the accuracy of methods become similar as we increase the size of training set, for small training sets GCD–2s (1D) and AMKL have the lowest misclassification error among methods that consider one-dimensional kernel parameters. In terms of training time, GCD–2s (1D) is the fastest method overall. Its training time is almost comparable to that of Uniform, which performs no kernel learning. Among multi-dimensional search methods GCD–2s (nD,NoReg) and GCD–2s (nD) perform better than CKL (nD). In particular, GCD–2s (nD,NoReg) is almost 10 times faster than its one-stage counterpart, CKL (nD).

In the second experiment, we again use the 50-dimensional synthetic dataset with $\gamma = 40$. We provide 50 training examples to all methods. One may argue that increasing the number of kernels provided to finite MKL algorithms will improve performance. Note that discretizing a multi-dimensional search space results in a combinatorial explosion in the number of kernels. However we can still increase the number of kernels over a one-dimensional space by using a finer grid and selecting more kernels from a fixed interval. In the second experiment we choose p Gaussian kernels, in a geometric manner, from the interval $[10^{-3}, 10^{3}]$, where $p \in \{50, 100, 200, 500, 1000, 2000\}$. We provide these kernels to finite MKL algorithms and measure misclassification error and training time. The results indicate

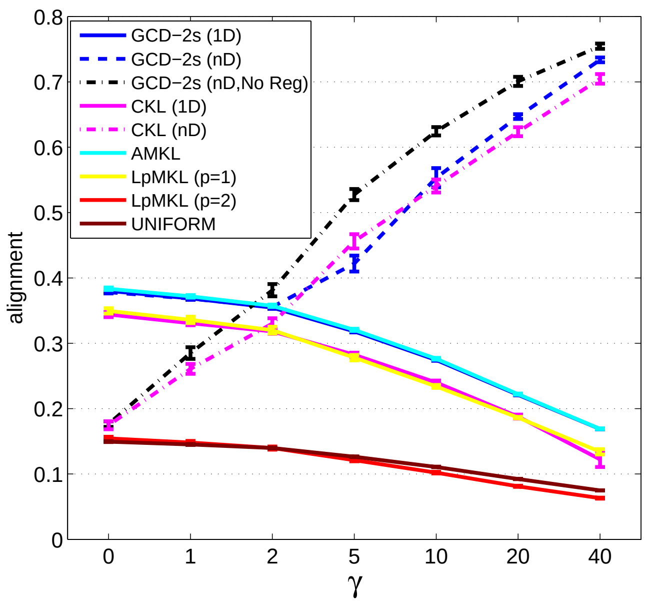

Figure 6.7: Alignment values in the 50-dimensional synthetic dataset experiment.

that increasing the number of base kernels do not improve performance. This is not unexpected: for $\gamma = 40$ many of the features are irrelevant and one-dimensional Gaussian kernels can not distinguish between relevant and irrelevant features. As a result, increasing the number of single-parameter kernels cannot help in this learning problem.

6.2.2 Gaussian kernels – real data

We evaluate several MKL methods on various binary classification tasks from MNIST and the UCI Letter recognition dataset, along with several other datasets from the UCI machine learning repository (Frank and Asuncion, 2010) and Delve datasets.

MNIST. In the first experiment, following Argyriou et al. (2005), we choose 8 handwritten digit recognition tasks of various difficulty from the MNIST dataset (LeCun and Cortes, 2010). This dataset consists of 28×28 images with pixel values ranging between 0 and 255. In these experiments, we aim to learn a combination of Gaussian kernels with one-dimensional parameter. Due to the large number of attributes (784) in the MNIST dataset, we do not evaluate multi-dimensional versions of GCD–2s and CKL (they are evaluated on a similar dataset, of smaller scale, see below). For finite MKL algorithms, we choose 20 kernels with the value

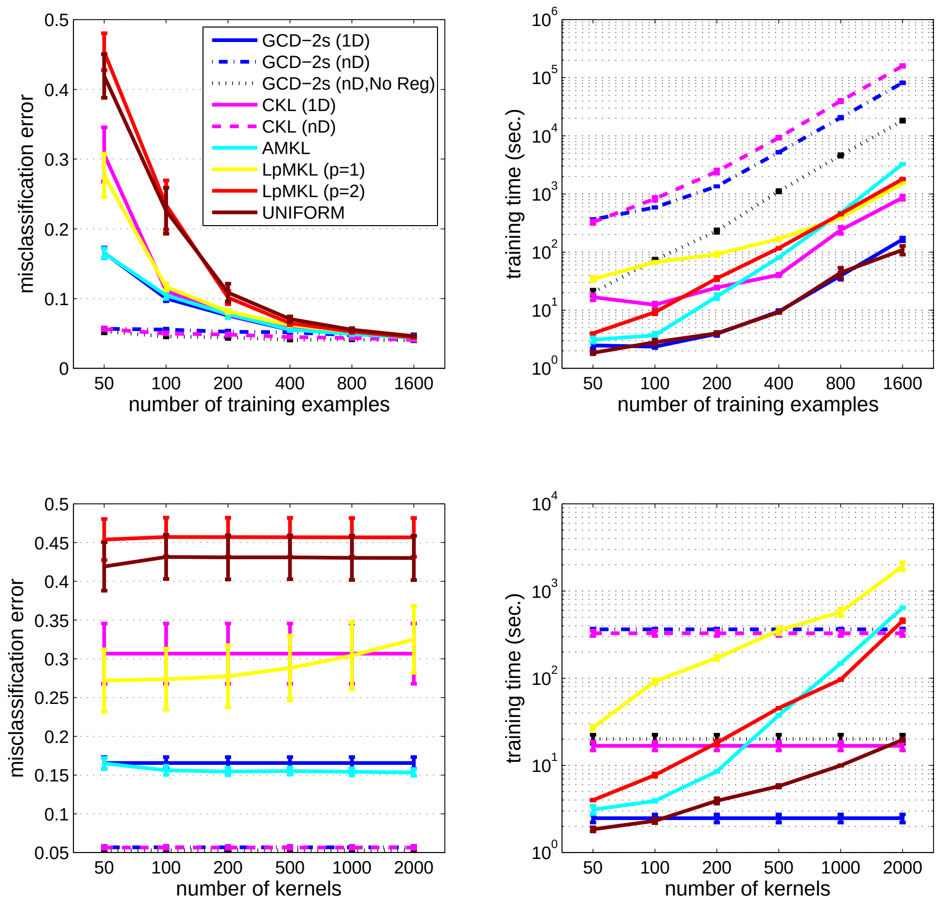

Figure 6.8: Scalability of various methods with respect to the number of training examples (top) and the number of kernels (bottom). Results for the methods that search over a continuous range are included for comparison purposes in the bottom figures. Note that the curves of these methods are flat as they do not need a finite set of kernels.

of σ picked from an equidistant discretization of interval $[500, 50000]$, following the methodology presented in Argyriou et al. (2005). In each experiment, the training and validation sets consist of 500 and 1000 data points, while the test set has 2000 data points. We repeated each experiment 10 times. The test-set error plots for all of the problems are shown in Figure 6.9. In order to give an overall impression of the algorithms' performance, we ranked them based on the results obtained in the above experiment. Table 6.4 reports median rank of each method for the experiment just described.

Overall, methods that choose σ from a continuous set outperformed their finite counterparts. This suggests again that for finite MKL methods the range of σ and

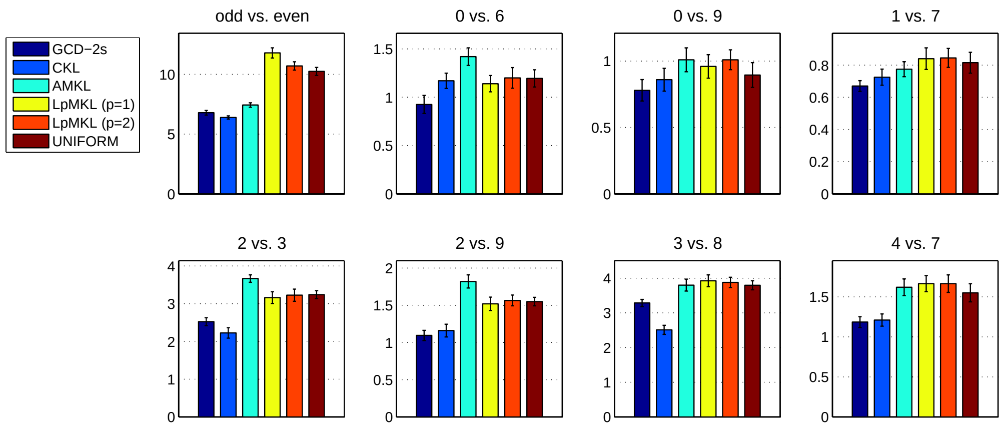

Figure 6.9: Misclassification percentages for different tasks from the MNIST dataset.

Table 6.4: Median rank and training time (seconds) of various kernel learning methods obtained in experiments.

	Rank			**Time**	
	MNIST	Letter	Other data	MNIST	Letter
GCD–2s (1D)	1	1	3	12 ± 1	9 ± 1
GCD–2s (nD)	N/A	5.5	2	N/A	770 ± 80
CKL (1D)	2	2	3	377 ± 56	590 ± 21
CKL (nD)	N/A	3	5	N/A	1550 ± 200
AMKL	4.5	4.5	3	31 ± 1	11 ± 1
LpMKL ($p = 1$)	4.5	8	4	57 ± 6	21 ± 1
LpMKL ($p = 2$)	5	7	7	58 ± 3	22 ± 1
Uniform	4	6	7	10 ± 1	5 ± 1

the discretization of this range is important to the accuracy of the resulting classifier.

UCI Letter Recognition. In another experiment, we evaluated these methods on 12 binary classification tasks from the UCI Letter recognition dataset. This dataset includes 20000 data points of the 26 capital letters in the English alphabet. For each binary classification task, the training and validation sets include 300 and 200 data points, respectively. The misclassification error is measured over 1000 test points. In this experiment, we run both the one- and n-dimensional search versions of GCD–2s and CKL. The rest of the methods learn a single parameter. Finite MKL methods were provided with 20 kernels with σ's chosen from the interval $[1, 200]$ in an equidistant manner. The number of kernels for these methods is

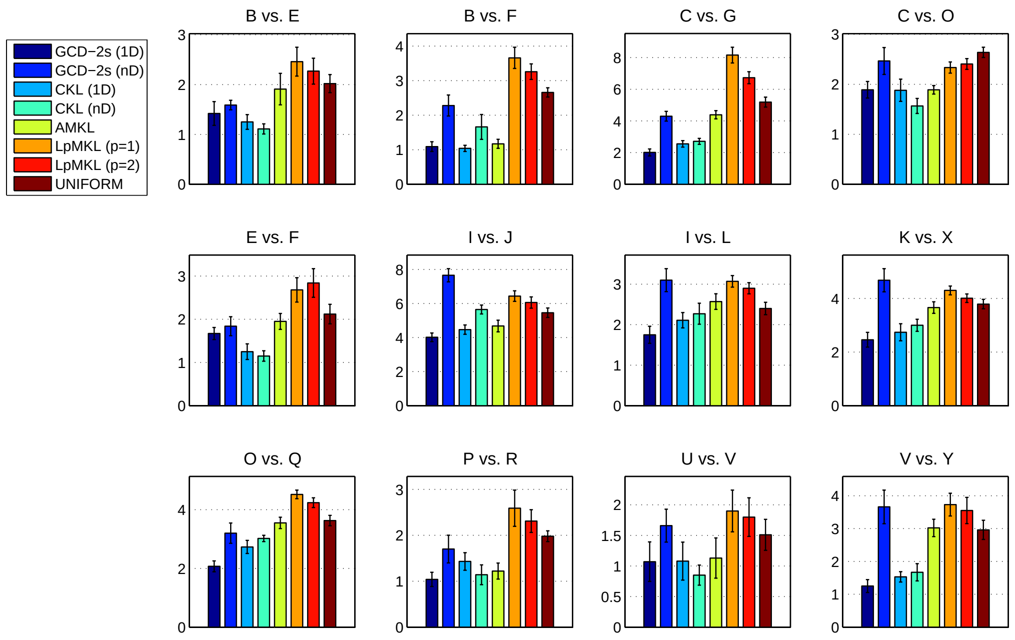

Figure 6.10: Misclassification percentages for different tasks from the UCI Letter recognition dataset.

chosen to make their training time comparable to that of GCD–2s (1D). The plots of misclassification error are shown in Figure 6.10. We report the median rank of each method in Table 6.4. While the one-dimensional version of our method outperforms the rest of the methods, the classifier built on the kernel found by the multi-dimensional version of our method did not perform well. To further investigate this discrepancy, we examined the value of alignment between the learned kernel and the target label kernel (ideal kernel) on the test set. The results are shown in Figure 6.11. The multi-dimensional version of our method achieved the highest value of alignment in every task in this experiment. This experiment demonstrates that *high alignment values between the learned kernel and the ideal kernel do not necessarily translate into a more accurate classifier.* Aside from this observation, the same trend observed in the MNIST data can be seen here. The continuous kernel learning methods outperform finite MKL methods here too.

Other datasets. In the last experiment we evaluate all methods on 11 datasets chosen from the UCI machine learning repository and Delve datasets. Most of these datasets were used previously to evaluate kernel learning algorithms (Lanckriet et al.,

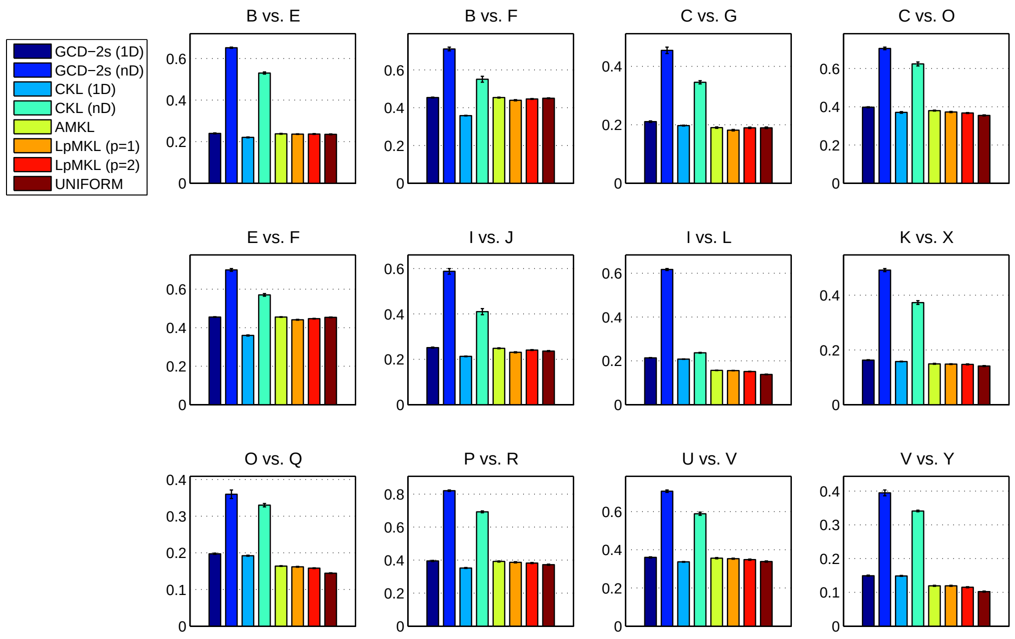

Figure 6.11: Alignment values in different tasks of the UCI Letter recognition dataset.

2004; Cortes et al., 2009a,b, 2010; Rakotomamonjy et al., 2008). The specification of these datasets are shown in Table 6.2. The performance of each method is shown in Figure 6.12. The median rank of each method is shown in Table 6.4. Contrary to the Letter experiment, in this case the multi-dimensional version of our method outperforms the rest of the methods.

Training Times. We measured the training time required for each run and each MKL method in the MNIST and the UCI Letter experiments. In each case, we took the average of training time for each method over all tasks. The average required time along with the standard error values are shown in Table 6.4. Among all methods, UNIFORM is fastest, which is expected, as it requires no additional time to compute kernel weights. The GCD–2S (1D) algorithm is the fastest among MKL methods. In these experiments our method converges in less than 10 iterations (kernels). The general trend is that one-stage MKL methods, i.e., LPMKL and CKL, are slower than two-stage methods, GCD–2S and AMKL. In these experiments CKL is slower than its counterpart, GCD–2S, since it usually requires more iterations (around 50) to converge.

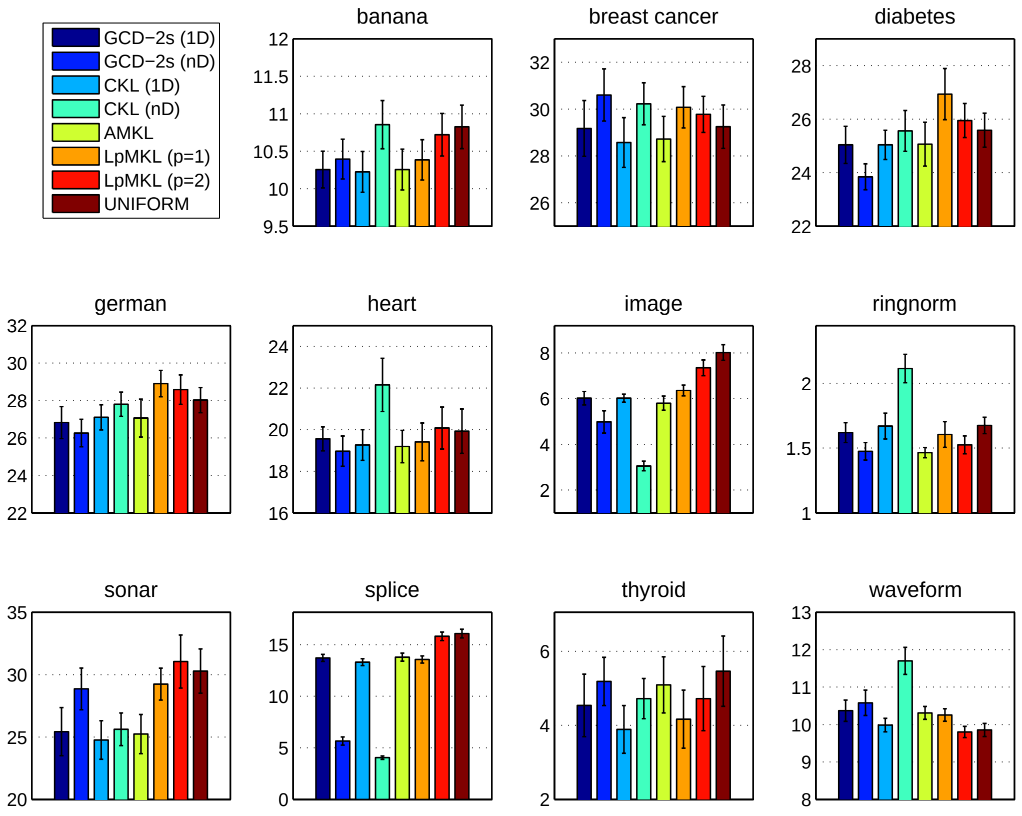

Figure 6.12: Misclassification percentages obtained in 11 datasets.

6.2.3 Non-convexity issue

In order to apply greedy coordinate descent to learn a combination of kernels that are continuously-parameterized, such as Gaussian kernels, one needs to solve an inner optimization problem in each iteration to find the best kernel parameter. This problem is not convex in general. The non-convexity issue exists for both one-stage and two-stage kernel learning (cf. problems (5.4) and (5.6) respectively). Here, we investigate this problem numerically, by plotting the function to be optimized in the case of a Gaussian kernel with a single bandwidth parameter when we apply GCD–2S algorithm. In particular, we plotted the objective function of problem (5.6) with its sign flipped, therefore we are interested in the local minima of function

$$h(\sigma) = -\left\langle \mathbf{K}(\kappa_\sigma), F'\left((\mathbf{K}(\kappa^{(k-1)}))_c \right) \right\rangle_F .$$

Figure 6.13 shows $h(\sigma)$ for several classification problems. The function h is shown for some iterations of some of the tasks from both the MNIST and the UCI Letter experiments. The number inside parentheses in the caption specifies the corresponding iteration of the algorithm. On these plots, the objective function does not have more than 2 local minima. Although in some cases the functions have some steep parts (at the scales shown), their optimization does not seem very difficult. It seems that non-convexity of inner optimization problem does not have a major influence on the performance of MKL algorithms, and simple solution, such as multiple restarts stated above, are sufficient to address this issue.

6.3 Stochastic Gradient Descent vs. Greedy Coordinate Descent for MKL

In this experiment, we compare stochastic gradient descent and greedy coordinate descent for one-stage and two-stage MKL. We consider learning a combination of Gaussian kernels.

We run GCD–1S, which is the same as CKL, SGD–1S, and GCD–2S with single- (1D), and multi-dimensional (nD) Gaussian kernels. Each bandwidth parameter is selected from the interval $[10^{-4}, 10^{4}]$. The regularization parameter is tuned from the set $\{10^{-4}, \ldots, 10^{3}\}$ using 5-fold cross validation. The stopping criterion for all methods is that the change in the value of the objective function in consecutive iterations is less than 10^{-3}. We found that no further improvement can

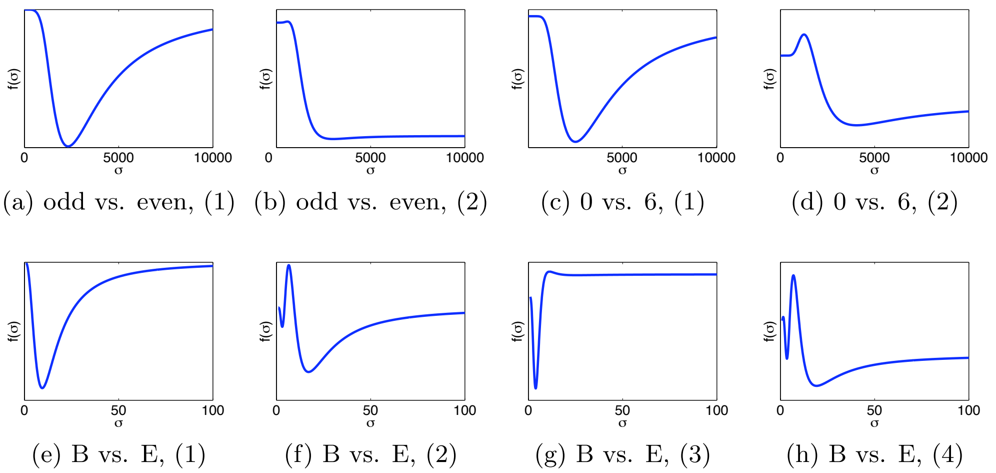

Figure 6.13: The flipped objective function underlying (5.6) as a function of σ, the parameter of a Gaussian kernel in selected MNIST and UCI Letter problems. Our algorithm needs to find the minimum of these functions (and similar ones).

be made by using smaller thresholds. We run each algorithm 10 times, where in each run the training and test data are chosen randomly. Error values are reported as mean squared error. The errors obtained from various algorithms are shown in Figure 6.14. To have an overall comparison, the median rank of these methods obtained over all datasets are also shown in Table 6.5. The general trend observed in this experiment is that alignment maximization could be beneficial in some cases. In fact, in this experiment, alignment maximizing two-stage MKL outperformed stochastic gradient descent and greedy coordinate descent for penalized empirical risk minimization. Another trend is combining Gaussian kernels with one parameter achieves better accuracy than combining multi-dimensional Gaussians. However, there are various cases such as diabetes, image, splice, and thyroid datasets, in which multi-dimensional kernel learning performs better than its one-dimensional counterpart. Stochastic gradient descent and greedy coordinate descent performed very similarly in this experiment. While they achieved the same rank in one-dimensional kernel parameter search, greedy coordinate descent performed slightly better in multi-dimensional parameter search.

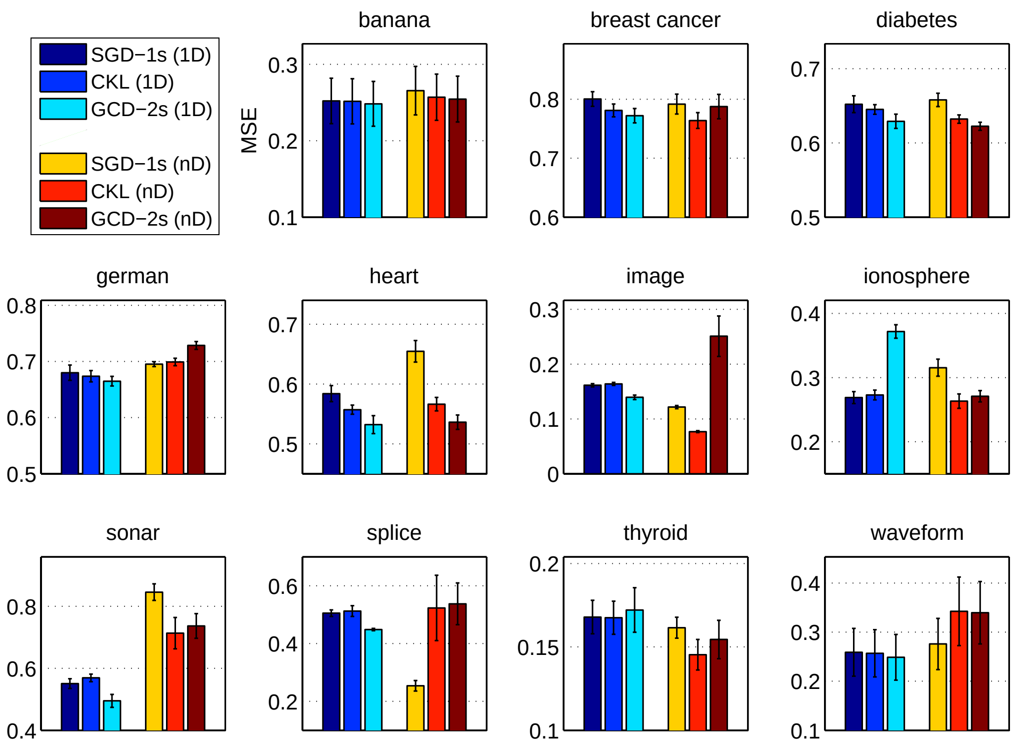

Figure 6.14: Prediction error of different methods in the Gaussian kernel experiment

6.4 Summary

In this chapter, we compared our stochastic gradient descent and greedy coordinate descent algorithms against several MKL algorithms. We showed that for stochastic gradient algorithms, sampling from a distribution that is proportional to the magnitude of gradient can be beneficial. We showed, using synthetic and real data, that our stochastic gradient descent MKL algorithm is comparable to state-of-the-art algorithms designed specifically to learn polynomial kernels. We then showed that the application of this algorithm is not limited to learning polynomial kernels. It can also be applied to learn a combination of infinitely-many kernels. In particular, we showed that it can be used to learn a combination of Gaussian kernels. We then illustrated some examples, in which, greedy coordinate descent was applied to one-stage and two-stage MKL. We showed that these methods are highly scalable and are comparable to state-of-the-art MKL algorithms.

An interesting observation from the experiments is that while alignment can be a good surrogate objective function in many applications, there are some cases in

Table 6.5: Median rank of methods in the Gaussian kernel experiment.

SGD–1s (1D)	CKL (1D)	GCD–2s (1D)	SGD–1s (nD)	CKL (nD)	GCD–2s (nD)
3	3	1.5	4.5	3.5	4

which using alignment might lead to overfitting.

Multiple kernel learning algorithms that we proposed in Chapters 4 and 5 are able to efficiently learn kernel combinations when the set of base kernels consists of exponentially or infinitely many kernels. This is in particular useful since standard MKL algorithms that are designed to combine a finite set of kernels are not scalable in such cases.

Chapter 7

Conclusion and Future Work

The problem of feature selection has been the focus of researchers in machine learning for many years. Multiple kernel learning offers a new approach to this problem by combining kernels, and hence, underlying feature sets, in the framework of learning with kernels. In Chapter 2, we examined two variants of problem formulation of MKL that have been widely used in the literature. In one-stage MKL, the kernel function and predictor are learned simultaneously through a penalized empirical risk minimization problem. In two-stage MKL, however, learning the kernel function and learning the predictor happen in two separate stages, and through separate optimization problems. We also showed that the solution to the problem of alignment maximization, which is not convex, can be obtained through solving an equivalent convex optimization problem.

Early MKL algorithms focus on learning linear combination of a given set of kernels through designing optimization problems for structural risk minimization. These works, however, are not scalable to large datasets or large number of kernels. Since then, most of the work has been devoted on developing faster algorithms that can deal with large scale data. While there have been some success in handling large datasets, there has not been much advance in the direction of developing algorithms that deal with large number of kernels. Examining the existing algorithms reveals that the computational complexity of most of the algorithms depends linearly on the number of kernels. Hence, for kernel sets with exponential or infinite number of kernels, these methods are not efficient.

In this thesis, we examined the shortcomings of current MKL algorithms and proposed new methods to handle large number of kernels. We showed in Chapter 3 that for continuously parameterized kernels, such as Dirichlet kernels or Gaussian kernels, the common practice of choosing base kernels by discretizing the parameter space may not achieve good performance if the discretization is not fine enough. This problem becomes more challenging when one deals with multi-dimensional kernel parameters as the number of kernels resulting from discretization increases exponentially. This calls for MKL algorithms that can select and combine kernels directly from the continuous space, or methods that can efficiently combine exponentially many kernels. Another important example of large kernel sets is polynomial kernels. Learning polynomial kernels requires a MKL algorithm to combine exponentially many kernels. Most of the current MKL algorithms are not scalable to handle such large kernel sets. Yet the problem of learning polynomial kernels is important as it enables one to perform non-linear feature selection.

In Chapter 4, we proposed a stochastic gradient descent algorithm for combining exponentially or infinitely many kernels. We also derived finite-time convergence guarantees for this algorithm. The main part of our algorithm, which makes it possible to handle large kernel sets, is to construct an unbiased randomized estimate of gradient with a controlled variance. We proposed a provably-correct sampling procedure for the important problem of learning polynomial kernels. The computational efficiency of this procedure comes from its logarithmic dependence on the size of the underlying kernel space. We also proposed efficient sampling procedures for Gaussian kernels.

Then, in Chapter 5, we developed a greedy coordinate descent algorithm for MKL. Greedy selection of coordinates is particularly useful when one considers continuously parameterized kernels since it becomes an optimization problem over kernel parameter. The algorithm of Argyriou et al. (2005) can be considered as a greedy coordinate descent algorithm for one-stage MKL. We proposed a similar algorithm for two-stage MKL. We also showed the connection of the new algorithm to the forward stagewise additive modeling framework.

The experimental results in Chapter 6 showed the efficiency of the new gradient-based algorithms. We compared these algorithms against several state-of-the-art MKL algorithms. We demonstrated that sampling from a distribution that is proportional to the magnitude of gradient results in a speed-up compared to other alternatives. The experiments on synthetic and real data showed that while standard MKL algorithms are not scalable to large kernel sets, the new algorithms are able to efficiently combine such kernel sets without any compromise in accuracy.

7.1 Future Work

In the following, we highlight several directions to extend this work.

Sampling procedure for Gaussian kernels

One of the key contributions in this thesis is a new kernel sampling procedure when the base kernels are tensor products of a given set of kernels (Section 4.2.1). While the number of such kernels is exponential, the complexity of sampling procedure is logarithmic in the number of base kernels. This results in an efficient stochastic gradient descent algorithm for learning polynomial kernels. We also showed that the variance of gradient estimates will remain bounded for the proposed sampling procedure. In Section 4.2.2, we proposed a sampling procedure for Gaussian kernels that samples kernels from a set with uncountably many kernels using an auxiliary distribution. While the proposed sampling procedure results in an efficient MKL algorithm with high generalization, the theoretical analysis required to guarantee the boundedness of variance of gradient estimate is left for future work.

Stochastic gradient descent for two-stage MKL

In Section 4.3, we discussed the possibility of applying stochastic gradient descent to two-stage MKL. However, we noticed that the gradient of the objective functions discussed, i.e. alignment and Euclidean distance, does not necessarily have all-positive or all-negative coordinates. Lack of this property makes it difficult to design an efficient sampling procedure for two-stage MKL with the above-mentioned objective functions. Note that we exploited this property to efficiently compute C_{k-1} for learning polynomial kernels in one-stage MKL. Two directions to overcome this problem are

1. Find suitable objective functions for two-stage MKL for which all coordinates of gradient vector have the same sign,
2. Design a new sampling procedure that does not require this condition.

Efficient algorithms for large datasets

We developed new MKL algorithms that can efficiently combine large sets of kernels. Yet the overall performance of algorithm depends also on the algorithm used to find the prediction function. This is in particular important for the new stochastic gradient descent algorithm proposed for MKL, since it requires the solution to optimization problem 4.9 in all iterations. One promising direction to address this problem is to use stochastic gradient descent to deal with large training data too. It has been shown that the run time of stochastic gradient descent does not depend on the number of training data, and in some cases, it even decreases as training set becomes larger (Bottou and Bousquet, 2008; Shalev-Shwartz and Srebro, 2008). In fact, using stochastic gradient descent to handle large datasets in MKL is not a new matter. Orabona et al. (2010) extended the stochastic gradient descent algorithm of (Shalev-Shwartz et al., 2007) to MKL. While their new algorithm can handle large number of training data, it is not efficient when it comes to combining large kernel sets as it updates all kernel coefficients in each iteration. We conjecture that stochastic gradient descent can be applied to efficiently handle large datasets and large number of kernels in MKL.

Appendix A

Proofs

In this section we present the proofs of Theorem 4.1, Proposition 4.2, and Theorem 5.1. The proof of Theorem 4.1 is based on the standard proof of the convergence rate of the mirror descent algorithm, see, for example, (Beck and Teboulle, 2003), or the proof of Proposition 2.2 of Nemirovski et al. (2009), which carry over the same argument to solve very similar but less general problems. We also provide some improvements and simplifications at the end. Before giving the actual proof, we need the following standard lemma:

Lemma A.1 (Lemma 2.1 of Nemirovski et al. 2009)**.** *Assume that Ψ is α-strongly convex with respect to some norm $\|\cdot\|$ (i.e., (4.1) holds). Let $\theta_1 \in K \cap A^\circ$, $\theta \in K \cap A$, and $g \in \mathbb{R}^d$. Define $\theta_2 = \arg\min_{\theta' \in K\cap A} \{\langle g, \theta' \rangle + D_\Psi(\theta', \theta_1)\}$. Then*

$$\langle g, \theta_1 - \theta \rangle \le D_\Psi(\theta, \theta_1) - D_\Psi(\theta, \theta_2) + \frac{\|g\|_*^2}{2\alpha}.$$

We provide an alternate proof that is based on the so-called 3-DIV lemma. The 3-DIV lemma (e.g., Lemma 11.1, Cesa-Bianchi and Lugosi, 2006) allows one to express the sum of the divergences between the vectors u, v and v, w in terms of the divergence between u and w and an additional "error term", where $u \in A$, $v, w \in A^\circ$:

$$D_\Psi(u, v) + D_\Psi(v, w) = D_\Psi(u, w) + \langle \nabla\Psi(w) - \nabla\Psi(v), u - v \rangle \ .$$

Proof. Note that $\theta_2 \in A^\circ$ due to behavior of Ψ at the boundary of A. Thus, Ψ is differentiable at θ_2 and

$$\nabla_1 D_\Psi(\theta_2, \theta_1) = \nabla\Psi(\theta_2) - \nabla\Psi(\theta_1)\,, \tag{A.1}$$

where ∇_1 denotes differentiation of D_Ψ w.r.t. its first variable. Let $f(\theta') = \langle g, \theta' \rangle + D_\Psi(\theta', \theta_1)$. By the optimality property of θ_2 and since $\theta \in K \cap A$, we have

$$\langle \nabla f(\theta_2), \theta_2 - \theta \rangle \le 0\,.$$

Plugging in the definition of f together with the identity (A.1) gives

$$\langle g + \nabla\Psi(\theta_2) - \nabla\Psi(\theta_1), \theta_2 - \theta \rangle \le 0\,. \tag{A.2}$$

Now, by the 3-DIV Lemma,

$$\begin{aligned} D_\Psi(\theta, \theta_2) + D_\Psi(\theta_2, \theta_1) &= D_\Psi(\theta, \theta_1) + \langle \nabla\Psi(\theta_1) - \nabla\Psi(\theta_2), \theta - \theta_2 \rangle \\ &= D_\Psi(\theta, \theta_1) + \langle g + \nabla\Psi(\theta_2) - \nabla\Psi(\theta_1), \theta_2 - \theta \rangle + \langle g, \theta - \theta_2 \rangle\,. \end{aligned}$$

Hence, by reordering and using the inequality (A.2) we get

$$\begin{aligned} D_\Psi(\theta, \theta_2) - D_\Psi(\theta, \theta_1) &\le \langle g, \theta - \theta_2 \rangle - D_\Psi(\theta_2, \theta_1) \\ &= \langle g, \theta_1 - \theta_2 \rangle - D_\Psi(\theta_2, \theta_1) + \langle g, \theta - \theta_1 \rangle \\ &\le \frac{\|g\|_*^2}{2\alpha} + \langle g, \theta - \theta_1 \rangle\ , \end{aligned}$$

where in the last line we used Young's inequality[1] and that due to the strong convexity of Ψ, $D_\Psi(\theta_2, \theta_1) \ge \frac{\alpha}{2}\|\theta_2 - \theta_1\|^2$. □

Theorem 4.1. *Assume that Ψ is α-strongly convex with respect to some norm $\|\cdot\|$ (with dual norm $\|\cdot\|_*$) for some $\alpha > 0$, that is, for any $\theta \in A^\circ, \theta' \in A$*

$$\Psi(\theta') - \Psi(\theta) \ge \left\langle \nabla\Psi(\theta), \theta' - \theta \right\rangle + \tfrac{\alpha}{2}\|\theta' - \theta\|^2. \tag{4.1}$$

Suppose, furthermore, that Algorithm 1 is run for T time steps. For $0 \le k \le T-1$ let $\mathcal{F}_k$ denote the σ-algebra generated by $\theta_1, \ldots, \theta_k$. Assume that, for all $1 \le k \le T$, $\hat{g}_k \in \mathbb{R}^d$ is an unbiased estimate of $\nabla J(\theta^{(k-1)})$ given $\mathcal{F}_{k-1}$, that is,

$$\mathbb{E}\left[\hat{g}_k \middle| \mathcal{F}_{k-1}\right] = \nabla J(\theta^{(k-1)}). \tag{4.2}$$

Further, assume that there exists a deterministic constant $B \ge 0$ such that for all $1 \le k \le T$,

$$\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1}\right] \le B \quad a.s. \tag{4.3}$$

[1] Young's inequality states that for any x, y vectors and $\alpha > 0$, $\langle x, y \rangle \le \|x\|_*\|y\| \le \frac{1}{2}\left(\frac{\|x\|_*^2}{\alpha} + \alpha\|y\|^2\right)$.

Finally, assume that $\delta = \sup_{\theta' \in K \cap A} \Psi(\theta') - \Psi(\theta^{(0)})$ *is finite. Then, if* $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{BT}}$ *for all* $k \geq 1$*, it holds that*

$$\mathbb{E}\left[J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right)\right] - \inf_{\theta \in K \cap A} J(\theta) \leq \sqrt{\frac{2B\delta}{\alpha T}}. \tag{4.4}$$

Furthermore, if

$$\|\hat{g}_k\|_*^2 \leq B' \quad a.s. \tag{4.5}$$

for some deterministic constant B' *and* $\eta_{k-1} = \sqrt{\frac{2\alpha\delta}{B'T}}$ *for all* $k \geq 1$ *then, for any* $0 < \varepsilon < 1$*, it holds with probability at least* $1 - \varepsilon$ *that*

$$J\left(\frac{1}{T}\sum_{k=1}^{T}\theta^{(k-1)}\right) - \inf_{\theta \in K \cap A} J(\theta) \leq \sqrt{\frac{2B'\delta}{\alpha T}} + 4\sqrt{\frac{B'\delta \log\frac{1}{\varepsilon}}{\alpha T}}. \tag{4.6}$$

Proof. Introduce the average learning rates $\overline{\eta}_{k-1}^{(T)} = \eta_{k-1}/\sum_{k=1}^{T}\eta_{k-1}$, $k = 1, \ldots, T$, the averaged parameter estimates

$$\bar{\theta}^{(T-1)} = \sum_{k=1}^{T}\overline{\eta}_{k-1}^{(T)}\theta^{(k-1)}$$

and choose some $\theta^* \in K \cap A$. To prove the first part of the theorem, it suffices to show that the bound holds for $J(\bar{\theta}^{(T-1)}) - J(\theta^*)$. Define $g_k = \nabla J\left(\theta^{(k-1)}\right)$. By the convexity of $J(\theta)$, we have

$$\begin{aligned} J\left(\bar{\theta}^{(T-1)}\right) - J(\theta^*) &\leq \sum_{k=1}^{T}\overline{\eta}_{k-1}^{(T)}\left(J\left(\theta^{(k-1)}\right) - J(\theta^*)\right) \\ &\leq \sum_{k=1}^{T}\overline{\eta}_{k-1}^{(T)}\left\langle g_k, \theta^{(k-1)} - \theta^*\right\rangle \\ &= \sum_{k=1}^{T}\overline{\eta}_{k-1}^{(T)}\left\langle \hat{g}_k, \theta^{(k-1)} - \theta^*\right\rangle \\ &+ \sum_{k=1}^{T}\overline{\eta}_{k-1}^{(T)}\left\langle g_k - \hat{g}_k, \theta^{(k-1)} - \theta^*\right\rangle \end{aligned} \tag{A.3}$$

Notice that the first term on the right hand side above is the sum of linearized losses appearing in the standard analysis of the mirror descent algorithm with loss functions $\hat{g}_k$ and learning rates $\overline{\eta}_{k-1}^{(T)}$, and the second sum contains the term that depends on how well $\hat{g}_k$ estimates the gradient g_k. Thus, in this way, it is separated how the mirror descent algorithm and the gradient estimate effect the convergence rate of the algorithm. The first sum can be bounded by invoking the standard bound for the mirror descent algorithm (we will give the very short proof for completeness, based on Lemma A.1), while the second sum can be analyzed by noticing that, by assumption (4.2), its elements form an $\{\mathcal{F}_k\}$-adapted martingale-difference sequence.

To bound the first sum, first note that the conditions of Lemma A.1 are satisfied for $\theta_1 = \theta^{(k-1)}, \theta = \theta^*, g = \overline{\eta}_{k-1}^{(T)}\hat{g}_k$, since $\theta_1 \in K \cap A^\circ$ (as mentioned beforehand, this follows from the behavior of Ψ at the boundary of A). Further, note that due to the so-called projection lemma (i.e., the D_Ψ-projection of the unconstrained optimizer is the same as the optimizer of the constrained optimization problem),we can conclude that $\theta^{(k)} = \theta_2$, where θ_2 is defined in Lemma A.1. Thus, Lemma A.1 gives

$$\eta_{k-1}\left\langle \hat{g}_k, \theta^{(k-1)} - \theta^* \right\rangle \le D_\Psi(\theta^*, \theta^{(k-1)}) - D_\Psi(\theta^*, \theta^{(k)}) + \frac{\eta_{k-1}^2 \|\hat{g}_k\|_*^2}{2\alpha}.$$

Summing the above inequality for $k = 1, \ldots, T$, the divergence terms cancel each other, yielding

$$\begin{aligned}&\sum_{k=1}^{T} \overline{\eta}_{k-1}^{(T)} \left\langle \hat{g}_k, \theta^{(k-1)} - \theta^* \right\rangle \\ \le\ & \frac{1}{\sum_{k=1}^T \eta_{k-1}} \left(D_\Psi(\theta^*, \theta^{(0)}) - D_\Psi(\theta^*, \theta^{(T)}) + \frac{1}{2\alpha}\sum_{k=1}^{T} \eta_{k-1}^2 \|\hat{g}_k\|_*^2 \right) .\end{aligned} \tag{A.4}$$

Let us now turn to the second sum. We start with developing a bound on the expected regret. For any $1 \le k \le T$, by construction $\overline{\eta}_{k-1}^{(T)}$ and $\theta^{(k-1)}$ are $\mathcal{F}_{k-1}$-measurable. This, together with (4.2) gives

$$\mathbb{E}\left[\overline{\eta}_{k-1}^{(T)} \left\langle g_k - \hat{g}_k, \theta^* - \theta^{(k-1)} \right\rangle \middle| \mathcal{F}_{k-1} \right] = \overline{\eta}_{k-1}^{(T)} \left\langle g_k - \mathbb{E}\left[\hat{g}_k \middle| \mathcal{F}_{k-1} \right], \theta^* - \theta^{(k-1)} \right\rangle = 0 \, . \tag{A.5}$$

Combining this result with (A.3) and (A.4) yields

$$\begin{aligned}&\mathbb{E}\left[J\left(\bar{\theta}^{(T)}\right) - J(\theta^*) \right] \\ \le\ & \frac{1}{\sum_{k=1}^T \eta_{k-1}} \left(D_\Psi(\theta^*, \theta^{(0)}) - D_\Psi(\theta^*, \theta^{(T)}) + \frac{1}{2\alpha}\sum_{k=1}^{T} \eta_{k-1}^2 \mathbb{E}\left[\mathbb{E}\left[\|\hat{g}_k\|_*^2 \middle| \mathcal{F}_{k-1} \right] \right] \right) \\ \le\ & \frac{\delta + \frac{1}{2\alpha}\sum_{k=1}^T \eta_{k-1}^2 B}{\sum_{k=1}^T \eta_{k-1}} \, ,\end{aligned} \tag{A.6}$$

where we used the tower rule to bring in the bound (4.3), the non-negativity of Bregman divergences, and $D_\Psi(\theta, \theta^{(0)}) \le \Psi(\theta) - \Psi(\theta^{(0)})$; the latter holds as

$$\left\langle \nabla\Psi(\theta^{(0)}), \theta - \theta^{(0)} \right\rangle \ge 0,$$

since $\theta^{(0)}$ minimizes Ψ on K. Substituting $\eta_{k-1} = \eta = \sqrt{\frac{2\alpha\delta}{BT}}, k = 1, \ldots, T$ finishes the proof of (4.4).

To prove the high probability result (4.6), notice that thanks to (4.2),

$$\left\{\eta_{k-1}\left\langle g_k - \hat{g}_k, \theta^* - \theta^{(k-1)}\right\rangle\right\}$$

is an $\{\mathcal{F}_k\}$-adapted martingale-difference sequence (cf. (A.5)). By the strong convexity of Ψ we have

$$\frac{\alpha}{2}\|\theta^{(k-1)} - \theta^*\|^2 \leq \Psi(\theta^{(k-1)}) - \Psi(\theta^*) \leq \delta.$$

Furthermore, conditions (4.2) and (4.5) imply that $\|g_k\|_*^2 \leq B'$ a.s., and so by (4.5) we have $\|g_k - \hat{g}_k\|_* \leq 2\sqrt{B'}$ a.s. Then by Hölder's inequality

$$\left|\left\langle g_k - \hat{g}_k, \theta^* - \theta^{(k-1)}\right\rangle\right| \leq \|g_k - \hat{g}_k\|_* \|\theta^* - \theta^{(k-1)}\| \leq 2\sqrt{\frac{2B'\delta}{\alpha}}.$$

Thus, by the Hoeffding-Azuma inequality (see, e.g., Lemma A.7, Cesa-Bianchi and Lugosi, 2006), for any $0 < \varepsilon < 1$ we have, with probability at least $1 - \varepsilon$,

$$\sum_{k=1}^{T} \overline{\eta}_{k-1}^{(T)}\left\langle g_k - \hat{g}_k, \theta^* - \theta^{(k-1)}\right\rangle \leq \frac{4}{\sum_{k=1}^{T}\eta_{k-1}}\sqrt{\frac{B'\delta}{\alpha}\left(\sum_{k=1}^{T}\eta_{k-1}^2\right)\ln\frac{1}{\varepsilon}}\,. \qquad \text{(A.7)}$$

Combining (A.4) with (4.5) implies an almost sure upper bound on the first sum on the right hand side of (A.3) as in (A.6) with B' in place of B. This, together with (A.7) proves the required high probability bound (4.6) when substituting $\eta_{k-1} = \eta' = \sqrt{\frac{2\alpha\delta}{B'T}}$.

□

Proposition 4.2. *For $t \in \{1, \ldots, n\}$, let $\ell_t^* : \mathbb{R} \to \mathbb{R}$ denote the convex conjugate of ℓ_t: $\ell_t^*(v) = \sup_{\tau\in\mathbb{R}}\{v\tau - \ell_t(\tau)\}$, $v \in \mathbb{R}$. For $i \in \mathcal{I}$, recall that $\kappa_i(x, x') = \langle\phi_i(x), \phi_i(x')\rangle$, and let $\mathbf{K}_i = (\kappa_i(x_t, x_s))_{t,s\in\{1,\ldots,n\}}$ be the $n \times n$ kernel matrix underlying κ_i and let $\mathbf{K}_\theta = \sum_{i\in\mathcal{I}}\frac{\theta_i}{\rho_i^2}\mathbf{K}_i$ be the kernel matrix underlying $\kappa_\theta = \sum_{i\in\mathcal{I}}\frac{\theta_i}{\rho_i^2}\kappa_i$. Then, for any fixed θ, the minimizer $w^*(\theta)$ of $J(\cdot, \theta)$ satisfies*

$$w_i^*(\theta) = \frac{\theta_i}{\rho_i^2}\sum_{t=1}^{n}\alpha_t^*(\theta)\phi_i(x_t), \quad i \in \mathcal{I}, \qquad (4.8)$$

where

$$\alpha^*(\theta) = \operatorname*{arg\,min}_{\alpha\in\mathbb{R}^n}\left\{\frac{1}{2}\alpha^\top\mathbf{K}_\theta\alpha + \frac{1}{n}\sum_{t=1}^{n}\ell_t^*(-n\alpha_t)\right\}. \qquad (4.9)$$

Proof. By introducing the variables $\tau = (\tau_t)_{1 \le t \le n} \in \mathbb{R}^n$ and using the definition of L we can write the optimization problem (2.9) as the constrained optimization problem

$$\min_{w \in \mathcal{W}, \tau \in \mathbb{R}^n} \frac{1}{n} \sum_{t=1}^{n} \ell_t(\tau_t) + \frac{1}{2} \sum_{i \in \mathcal{I}} \frac{\rho_i^2 \|w_i\|_2^2}{\theta_i} \qquad \text{s.t. } \tau_t = \sum_{i \in \mathcal{I}} \langle w_i, \phi_i(x_t) \rangle , \tag{A.8}$$

In what follows, we call this problem the primal problem. The Lagrangian of this problem is

$$\mathcal{L}(w, \tau, \alpha) \overset{\text{def}}{=} \frac{1}{n} \sum_{t=1}^{n} \ell_t(\tau_t) + \frac{1}{2} \sum_{i \in \mathcal{I}} \frac{\rho_i^2 \|w_i\|_2^2}{\theta_i} + \sum_{t=1}^{n} \alpha_t \left\{ \tau_t - \sum_{i \in \mathcal{I}} \langle w_i, \phi_i(x_t) \rangle \right\} ,$$

where $\alpha = (\alpha_t)_{1 \le t \le n} \in \mathbb{R}^n$ is the vector of Lagrange multipliers (or dual variables) associated with the n equality constraints. The Lagrange dual function, $g(\alpha) \overset{\text{def}}{=} \inf_{w,\tau} \mathcal{L}(w, \tau, \alpha)$, can be readily seen to satisfy

$$g(\alpha) = -\left(\frac{1}{2} \alpha^\top \mathbf{K}_\theta \alpha + \frac{1}{n} \sum_{t=1}^{n} \ell_t^*(-n\alpha_t) \right) .$$

Now, since the objective function of the primal problem is convex and the primal problem involves only affine equality constraints and the primal problem is clearly feasible, by Slater's condition (p.226, Boyd and Vandenberghe, 2004), if $\alpha^*(\theta)$ is the maximizer of $g(\alpha)$ then

$$\begin{aligned} w^*(\theta) &= \operatorname*{arg\,min}_{w \in \mathcal{W}} \inf_{\tau \in \mathbb{R}^n} \mathcal{L}(w, \tau, \alpha^*(\theta)) \\ &= \operatorname*{arg\,min}_{w \in \mathcal{W}} \sum_{i \in \mathcal{I}} \left\{ \frac{\rho_i^2 \|w_i\|_2^2}{2\theta_i} - \sum_{t=1}^{n} \alpha_t \langle w_i, \phi_i(x_t) \rangle \right\} . \end{aligned}$$

The minimum of the last expression is readily seen to be equal to the expression given in (4.8), thus finishing the proof. □

Theorem 5.1. *For each $k \ge 1$, the sequence $\theta^{(k)}$ obtained by Algorithm 5 satisfy*

$$J(\theta^{(k)}) - J(\theta^*) \le \frac{2C_J}{k+2}, \tag{5.2}$$

where θ^* is the minimizer of function J over simplex Δ_1, J is assumed to be convex and differentiable, where for all $x, z \in \Delta_1$ and $\eta \in [0, 1]$ it satisfies

$$J((1-\eta)x + \eta z) \le J(x) + \eta \langle \nabla J(x), z - x \rangle + \frac{C_J}{2} \eta^2,$$

where C_J is a measure of curvature of J.

Proof. We prove Theorem 5.1 for the case that the learning rates are set to $\eta_k = \frac{2}{k+1}$:

$$\begin{aligned}
J(\theta^{(k)}) &= J((1-\eta_k)\theta^{(k-1)} + \eta_k e_I) \\
\text{(assumption on } J) &\leq J(\theta^{(k-1)}) + \eta_k \left\langle \nabla J(\theta^{(k-1)}), e_I - \theta^{(k-1)} \right\rangle + \frac{C_J}{2}\eta_k^2 \\
\text{(line 6 in Algorithm 5)} &\leq J(\theta^{(k-1)}) + \eta_k \left\langle \nabla J(\theta^{(k-1)}), \theta^* - \theta^{(k-1)} \right\rangle + \frac{C_J}{2}\eta_k^2 \\
\text{(convexity of } J) &\leq J(\theta^{(k-1)}) + \eta_k (J(\theta^*) - J(\theta^{(k-1)})) + \frac{C_J}{2}\eta_k^2 \\
&= (1-\eta_k) J(\theta^{(k-1)}) + \eta_k J(\theta^*) + \frac{C_J}{2}\eta_k^2.
\end{aligned}$$

Adding $-J(\theta^*)$ to both sides yields

$$J(\theta^{(k)}) - J(\theta^*) \leq (1-\eta_k)(J(\theta^{(k-1)}) - J(\theta^*)) + \frac{C_J}{2}\eta_k^2.$$

Let $h(\theta^{(k)}) = J(\theta^{(k)}) - J(\theta^*)$ and $C = C_J/2$. We have

$$h(\theta^{(k)}) \leq (1-\eta_k) h(\theta^{(k-1)}) + C\eta_k^2. \tag{A.9}$$

We show by induction that

$$h(\theta^{(k)}) \leq \frac{4C}{k+2}.$$

For $k = 1$, it is obvious from (A.9) that $h(\theta^{(1)}) \leq C \leq 4C/3$. It remains to show that

$$h(\theta^{(k+1)}) \leq \frac{4C}{k+3}.$$

From (A.9) we have

$$\begin{aligned}
h(\theta^{(k+1)}) &\leq (1 - \frac{2}{k+2}) h(\theta^{(k)}) + \frac{4C}{(k+2)^2} \\
&\leq (1 - \frac{2}{k+2}) \frac{4C}{k+2} + \frac{4C}{(k+2)^2} \\
&\leq \frac{4C}{k+3},
\end{aligned}$$

which concludes the proof. Note that optimizing η_k through line search does not alter the proof and will only improve the speed of convergence of the algorithm. □

Appendix B

Derivations

In this section we show how the value of the scaling factor in the problem of Euclidean distance minimization can be calculated to ensure that $\theta^* \in \Delta_1$. In the following analysis, we consider the more general case where each kernel κ_j has a scaling factor β_j. The case of a single scaling factor simply follows by setting $\beta = \max_{i\in\mathcal{I}} \beta_i$. In what follows, for simplicity of notation we write $\mathbf{K}^*$ instead of $\hat{\mathbf{K}}_c^*$.

Calculating the scaling factor

To ensure that $\theta^* \in \Delta_1$, note that the gradient along each direction must be non-negative on the boundary of the simplex Δ_1:

$$\frac{\partial}{\partial\theta_i} J(\theta) = \frac{1}{n^2}\left\langle \beta_i \mathbf{K}_i, \sum_{j\in\mathcal{I}} \theta_j \beta_j \mathbf{K}_j - \mathbf{K}^* \right\rangle_F \geq 0, \quad \text{if } \sum_{k\in\mathcal{I}} \theta_k = 1.$$

Divide by $\beta_i/n^2 > 0$, and rearrange:

$$\sum_{j\in\mathcal{I}} \theta_j \beta_j \left\langle \mathbf{K}_i, \mathbf{K}_j \right\rangle_F \geq \left\langle \mathbf{K}_i, \mathbf{K}^* \right\rangle$$

Divide by $\left\langle \mathbf{K}_i, \mathbf{K}^* \right\rangle_F$:

$$\sum_{j\in\mathcal{I}} \theta_j \beta_j \rho_{ij} \geq 1, \quad \forall i \in \mathcal{I},$$

where $\rho_{ij} = \left\langle \mathbf{K}_i, \mathbf{K}_j \right\rangle_F / \left\langle \mathbf{K}_i, \mathbf{K}^* \right\rangle_F > 0$. To ensure that it holds for all $i \in \mathcal{I}$, it suffices to show

$$\sum_{j\in\mathcal{I}} \theta_j \beta_j \min_{i\in\mathcal{I}} \rho_{ij} \geq 1.$$

Since $\sum_{j\in\mathcal{I}} \theta_j = 1$, it suffices that

$$\begin{aligned}
&\min_{j\in\mathcal{I}} \beta_j \min_{i\in\mathcal{I}} \rho_{ij} \geq 1, \\
\iff\quad & \beta_j \min_{i\in\mathcal{I}} \rho_{ij} \geq 1, \quad \forall j \in \mathcal{I} \\
\iff\quad & \beta_j \geq \frac{1}{\min_{i\in\mathcal{I}} \rho_{ij}}, \quad \forall j \in \mathcal{I} \\
\iff\quad & \beta_j \geq \max_{i\in\mathcal{I}} \frac{\langle \mathbf{K}_i, \mathbf{K}^* \rangle_F}{\langle \mathbf{K}_i, \mathbf{K}_j \rangle_F}, \quad \forall j \in \mathcal{I}.
\end{aligned} \tag{B.1}$$

As an example, we consider Gaussian kernels of the form

$$\mathbf{K}_\sigma(x_i, x_j) = \exp(-\frac{D_{ij}}{\sigma^2}), \quad \sigma \in \Sigma$$

where

$$D_{ij} = \|x_i - x_j\|_2^2,$$

and Σ is a bounded interval. We calculate two different values for β to ensure that $\theta^* \in \Delta_1$. For continuously-parameterized Gaussian kernels condition (B.1) becomes

$$\sup_{\sigma\in\Sigma} \frac{\sum_{i,j=1}^n \mathbf{K}^*_{ij} \exp(-D_{ij}/\sigma^2)}{\sum_{i,j=1}^n \exp(-D_{ij}/\sigma^2) \exp(-D_{ij}/\delta^2)}, \quad \delta \in \Sigma.$$

The numerator can be upper-bounded by

$$\sum_{i,j=1}^n \mathbf{K}^*_{ij} \underbrace{\exp(-D_{ij}/\sigma^2)}_{\geq 0} \leq \max_{ij} \mathbf{K}^*_{ij} \sum_{ij} \exp(-D_{ij}/\sigma^2).$$

The denominator can be lower-bounded by

$$\sum_{i,j=1}^n \exp(-D_{ij}/\sigma^2) \exp(-D_{ij}/\delta^2) \geq \exp(\Delta_M/\delta_m^2) \sum_{i,j=1}^n \exp(-D_{ij}/\sigma^2),$$

where $D_M = \max_{ij} D_{ij}$, and δ_m is the smallest member of Σ. Hence, β can be chosen to be

$$\beta = \frac{\max_{ij} \mathbf{K}^*_{ij}}{\exp(D_M/\delta_m^2)}. \tag{B.2}$$

The expression given for β in (B.2) does not depend on the number of data points. However, in practice it may be a large value. Another expression for β can be obtained by noting that the denominator can be written as

$$n + \sum_{i\neq j} \exp(-\frac{D_{ij}}{\sigma^2}) \exp(-\frac{D_{ij}}{\delta^2}),$$

since $D_{ii} = 0$, $\forall i$. Now, to lower-bound the right-hand side we choose $\delta \to 0$, hence the denominator can be lower-bounded by n. Also, the numerator can be upper-bounded by

$$\sum_{i,j=1}^{n} \mathbf{K}^*_{ij} \underbrace{\exp(-D_{ij}/\sigma^2)}_{\geq 0} \leq \max_{ij} \mathbf{K}^*_{ij} \sum_{ij} \underbrace{\exp(-D_{ij}/\sigma^2)}_{\leq 1} \leq \max_{ij} \mathbf{K}^*_{ij} n^2$$

Note that for a centered kernel $\mathbf{K}^*$ we have $\sum_{i,j=1}^{n} \mathbf{K}^*_{ij} = 0$. Therefore, the above upper bound is loose. Nonetheless, the β can be chosen to be

$$\beta = n \max_{ij} \mathbf{K}^*_{ij}. \tag{B.3}$$

Effect of the number of dimensions of θ on the curvature parameter C_J

It can be shown that for the loss function in (5.9) the curvature parameter C_J does not depend on the number of dimensions of θ. For this loss function we have

$$\nabla^2 J(\theta) = \frac{\beta}{n^2} M,$$

where

$$M_{ij} = \langle \mathbf{K}_i, \mathbf{K}_j \rangle_F \geq 0, \quad i, j \in \mathcal{I}.$$

Hence, for the bound to hold, the parameter C_J can be chosen to be

$$C_J = \sup_{X,Y \in \Delta_1} \frac{\beta}{n^2} (X - Y)^\top M (X - Y),$$

where Δ_1 is the unit simplex in $\mathbb{R}^{|\mathcal{I}|}$. If $\mathbf{K}_i(X, X') \leq 1$, $\forall i \in \mathcal{I}$, which holds for instance for Gaussian kernels, we have

$$M_{ij} \leq n^2 \quad \forall i, j \in \mathcal{I}.$$

Hence,

$$\begin{aligned} \frac{\beta}{n^2}(X - Y)^\top M (X - Y) &\leq \beta (X - Y)^\top \mathbf{1}\mathbf{1}^\top (X - Y) \\ &\leq \beta \|X - Y\|_1^2 \\ &\leq 4\beta, \end{aligned}$$

where the last inequality holds since $X, Y \in \Delta_1$.

Bibliography

Argyriou, A., Hauser, R., Micchelli, C., and Pontil, M. (2006). A DC-programming algorithm for kernel selection. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 41–48.

Argyriou, A., Micchelli, C., and Pontil, M. (2005). Learning convex combinations of continuously parameterized basic kernels. In *Proceedings of the 18th Annual Conference on Learning Theory*, pages 338–352.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68(3):337–404.

Bach, F. (2008). Exploring large feature spaces with hierarchical multiple kernel learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 105–112.

Beck, A. and Teboulle, M. (2003). Mirror descent and nonlinear projected subgradient methods for convex optimization. *Operations Research Letters*, 31(3):167–175.

Bottou, L. and Bousquet, O. (2008). The tradeoffs of large scale learning. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 161–168. MIT Press, Cambridge, MA.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization.* Cambridge University Press.

Brown, A. and Page, A. (1970). *Elements of Functional Analysis.* Van Nostrand Reinhold Company, Windsor House, 46 Victoria Street, London S .W.1, England.

Bühlmann, P. and van de Geer, S. (2011). *Statistics for High-Dimensional Data: Methods, Theory and Applications.* Springer.

Cesa-Bianchi, N. and Lugosi, G. (2006). *Prediction, Learning, and Games.* Cambridge University Press, New York, NY, USA.

Chapelle, O. and Rakotomamonjy, A. (2008). Second order optimization of kernel parameters. In *Proc. of the NIPS Workshop on Kernel Learning: Automatic Selection of Optimal Kernels.*

Clarkson, K. L. (2008). Coresets, sparse greedy approximation, and the frank-wolfe algorithm. In *SODA*, pages 922–931.

Cortes, C., Mohri, M., and Rostamizadeh, A. (2009a). L2 regularization for learning kernels. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 109–116.

Cortes, C., Mohri, M., and Rostamizadeh, A. (2009b). Learning non-linear combinations of kernels. In *Advances in Neural Information Processing Systems*, volume 22, pages 396–404.

Cortes, C., Mohri, M., and Rostamizadeh, A. (2010). Two-stage learning kernel algorithms. In *Proceedings of the 27th International Conference on Machine Learning*, pages 239–246.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3):273–297.

Cristianini, N., Kandola, J., Elisseeff, A., and Shawe-Taylor, J. (2002). On kernel-target alignment. In *Advances in Neural Information Processing Systems*, volume 15, pages 367–373.

Do, C., Le, Q., and Foo, C. (2009). Proximal regularization for online and batch learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 257–264. ACM.

Frank, A. and Asuncion, A. (2010). UCI machine learning repository.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval research logistics quarterly*, 3:95–110.

Gehler, P. and Nowozin, S. (2008). Infinite kernel learning. Technical Report 178, Max Planck Institute For Biological Cybernetics.

Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer-Verlag New York.

Hazan, E., Agarwal, A., and Kale, S. (2007). Logarithmic regret algorithms for online convex optimization. *Machine Learning Journal*, 69(2-3):169–192.

Hazan, E. and Kale, S. (2011). Beyond the regret minimization barrier: an optimal algorithm for stochastic strongly-convex optimization. In *Proceedings of the 24th Annual Conference on Learning Theory*, volume 19 of *JMLR Workshop and Conference Proceedings*, pages 421–436.

He, J., Chang, S., and Xie, L. (2008). Fast kernel learning for spatial pyramid matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–7. IEEE.

Igel, C., Glasmachers, T., Mersch, B., Pfeifer, N., and Meinicke, P. (2007). Gradient-based optimization of kernel-target alignment for sequence kernels applied to bacterial gene start detection. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 4(2):216–226.

Jaggi, M. (2013). Revisiting frank-wolfe: Projection-free sparse convex optimization. ICML.

Kandola, J., Shawe-Taylor, J., and Cristianini, N. (2002). Optimizing kernel alignment over combinations of kernel.

Kimeldorf, G. and Wahba, G. (1971). Some results on tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33(1):82–95.

Kloft, M., Brefeld, U., Sonnenburg, S., and Zien, A. (2011). l_p-norm multiple kernel learning. *Journal of Machine Learning Research*, 12:953–997.

Lanckriet, G., Cristianini, N., Bartlett, P., Ghaoui, L., and Jordan, M. (2004). Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Micchelli, C. and Pontil, M. (2005). Learning the kernel function via regularization. *Journal of Machine Learning Research*, 6:1099–1125.

Nemirovski, A., Juditsky, A., Lan, G., and Shapiro, A. (2009). Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609.

Nemirovski, A. and Yudin, D. (1998). *Problem Complexity and Method Efficiency in Optimization*. Wiley.

Nesterov, Y. (2010). Efficiency of coordinate descent methods on huge-scale optimization problems. *CORE Discussion paper*, (2010/2).

Nesterov, Y. (2012). Subgradient methods for huge-scale optimization problems. *CORE Discussion paper*, (2012/2).

Nguyen, C. and Ho, T. (2008). An efficient kernel matrix evaluation measure. *Pattern Recognition*, 41(11):3366–3372.

Orabona, F. and Jie, L. (2011). Ultra-fast optimization algorithm for sparse multi kernel learning. In *International Conference on Machine Learning (ICML-11)*, pages 249–256.

Orabona, F., Jie, L., and Caputo, B. (2010). Online-batch strongly convex multi kernel learning. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 787–794. IEEE.

Rakotomamonjy, A., Bach, F., Canu, S., and Grandvalet, Y. (2008). SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521.

Richtárik, P. and Takáĉ, M. (2011). Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. (revised July 4, 2011) submitted to Mathematical Programming.

Rockafellar, R. (1976). Monotone operators and the proximal point algorithm. *SIAM Journal on Control and Optimization*, 14(1):877–898.

Schölkopf, B. and Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th international conference on Machine learning*, pages 807–814. ACM.

Shalev-Shwartz, S. and Srebro, N. (2008). Svm optimization: inverse dependence on training set size. In *Proceedings of the 25th international conference on Machine learning*, pages 928–935. ACM.

Shalev-Shwartz, S. and Tewari, A. (2011). Stochastic methods for l_1-regularized loss minimization. *Journal of Machine Learning Research*, 12:1865–1892.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel Methods for Pattern Analysis*. Cambridge Univ Press.

Sonnenburg, S., Rätsch, G., Schäfer, C., and Schölkopf, B. (2006). Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565.

Steinwart, I. and Christmann, A. (2008). *Support vector machines*. Springer.

Tanabe, H., Ho, T., Nguyen, C., and Kawasaki, S. (2008). Simple but effective methods for combining kernels in computational biology. In *Research, Innovation and Vision for the Future, 2008. RIVF 2008. IEEE International Conference on*, pages 71–78. IEEE.

Tomioka, R. and Suzuki, T. (2010). Sparsity-accuracy trade-off in mkl. *arXiv preprint arXiv:1001.2615*.

Xu, Z., Jin, R., King, I., and Lyu, M. (2008). An extended level method for efficient multiple kernel learning. In *Advances in Neural Information Processing Systems*, volume 21, pages 1825–1832.

Xu, Z., Jin, R., Yang, H., King, I., and Lyu, M. (2010). Simple and efficient multiple kernel learning by group lasso. In *Proceedings of the 27th International Conference on Machine Learning*, pages 1175–1182.

Ying, Y., Huang, K., and Campbell, C. (2009). Enhanced protein fold recognition through a novel data integration approach. *BMC bioinformatics*, 10(1):267.

Zhang, T. (2003). Sequential greedy approximation for certain convex optimization problems. *Information Theory, IEEE Transactions on*, 49(3):682–691.

Zou, H. and Hastie, T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.