Feature Selection and Kernel Design via Linear Programming

Glenn Fung, Romer Rosales, R. Bharat Rao
Siemens Medical Solutions, 51 Valley Stream Parkway, Malvern, PA, USA.
glenn.fung@siemens.com, romer.rosales@siemens.com, bharat.rao@siemens.com

Abstract

The definition of object (*e.g.,* data point) *similarity* is critical to the performance of many machine learning algorithms, both in terms of accuracy and computational efficiency. However, it is often the case that a similarity function is unknown or chosen by hand. This paper introduces a formulation that given relative similarity comparisons among triples of points of the form *object i is more like object j than object k,* it constructs a kernel function that preserves the given relationships. Our approach is based on learning a kernel that is a combination of functions taken from a set of base functions (these could be kernels as well). The formulation is based on defining an optimization problem that can be solved using linear programming instead of a semi-definite program usually required for kernel learning. We show how to construct a convex problem from the given set of similarity comparisons and then arrive to a linear programming formulation by employing a subset of the positive definite matrices. We extend this formulation to consider representation/evaluation efficiency based on formulating a novel form of feature selection using kernels (that is not much more expensive to solve). Using publicly available data, we experimentally demonstrate how the formulation introduced in this paper shows excellent performance in practice by comparing it with a baseline method and a related state-of-the art approach, in addition of being much more efficient computationally.

1 Introduction and Related Work

The definition of a *distance* or a *similarity function* over the input or data space is fundamental in the performance of machine learning algorithms, both in terms of accuracy and efficiency. This can be easily verified by looking at the role that the distance or similarity plays in common algorithms (*e.g.,* k-means, nearest neighbors, support vector machines or any other kernel method, etc.). However, since the concept of similarity depends on the task of interest (objects can be similar or dissimilar in many ways depending on the application), a similarity function that is appropriate for one task may not be appropriate for another. This task dependency has been noted in a number of approaches that seek to build these functions for a particular classification or regression model given some training data (*e.g.,* [Cohen *et al.*, 1998; Xing *et al.*, 2002; Schultz and Joachims, 2003; Lanckriet *et al.*, 2004; Athitsos *et al.*, 2004]).

In this paper, we are interested in automatically finding *good* similarity functions after some basic information about the task of interest has been provided (*e.g.,* by an expert user). Unlike approaches that rely on class labels (*e.g.,* [Lanckriet *et al.*, 2004]) or on building sets of points that are globally *similar* (or *dissimilar*) [Xing *et al.*, 2002; Wagstaff *et al.*, 2001], here we explore a simpler-to-specify form of user supervision: that provided by statements like *object i is more similar to object j than to object k*[1]. We remark that we are not only interested on building functions that are consistent with this information provided by the user, but we are also interested in obtaining functions that are efficient to compute. Thus, in addition we focus on functions that can be evaluated by looking only at part of the features or input dimensions; therefore, implying some form of feature selection.

More formally, let us represent the objects of interest as points $\mathbf{x}_k$ in a D-dimensional space $\Re^D$, with $k = \{1, 2, ..., N\}$. We would like to obtain a function $K : \Re^D \times \Re^D \rightarrow \Re$ that satisfies the above similarity comparisons and that in addition can be well approximated by $\tilde{K} : \Re^d \times \Re^d \rightarrow \Re$ which only uses $d < D$ components of $\mathbf{x}_k$. For this we will rely on (Mercer) Kernel representations [Cristianini and Shawe-Taylor, 2000] and define K (similarly for $\tilde{K}$) as a kernel $K(\mathbf{x}, \mathbf{y}) = \sum_j \alpha_j K_j(\mathbf{x}, \mathbf{y})$ where $\alpha \in \Re$; thus, K is a mixture of kernel functions[2]. Throughout this paper, we will work with this representation of similarity functions, define convex problems for obtaining K, and later extend this idea to focus on $\tilde{K}$. Some of the motivation for the framework in this paper is related to earlier work in metric learning [Rosales and Fung, 2006]; however, here the focus is on a different problem, kernel design.

[1]Example objects of interest include database records, user opinions, product characteristics, etc.

[2]Note that K is any affine combination of kernels not just a convex combination. Also, we are not restricting K_j to be itself a kernel.

In this paper, we first show how relative constraints, involving triples of points of the form, $K(\mathbf{x}_i, \mathbf{x}_j) > K(\mathbf{x}_i, \mathbf{x}_k)$ can be used to learn K. One can think of this first formulation as parallel to other approaches for learning K that rely on supervised learning with class labels (and, unlike the one presented here, are classification-algorithm dependent). These formulations involve solving a semidefinite programming problem (SDP) [Graepel, 2002; Lanckriet *et al.*, 2004], but we show how a different Linear Programming formulation, introduced in this paper, is also sufficient in practice and much more efficient computationally (*e.g.,* it can be use to solve much larger problems, faster). In addition, we extend this formulation to consider the issue of representation/evaluation efficiency based on a form of feature selection. This formulation leads to a linear programming approach for solving the kernel design problem that also optimizes for succinct feature representations.

1.1 Kernel Design

In recent years the idea of kernel design for learning algorithms has received considerable attention in the machine learning community. Traditionally, an appropriate kernel is found by choosing a parametric family of well-known kernels, *e.g.,* Gaussian or polynomial, and then learning a generally sub-optimal set of parameters by a tuning procedure. This procedure, although effective in most cases, suffers from important drawbacks: (1) calculating kernel matrices for a large range of parameters may become computationally prohibitive, especially when the training data is large and (2) very little (to none) prior knowledge about the desired similarity can be easily incorporated into the kernel learning procedure.

More recently, several authors [Bennett *et al.*, 2002; Lanckriet *et al.*, 2004; Fung *et al.*, 2004] have considered the use of a linear combination of kernels that belong to a family or superset of different kernel functions and parameters; this transforms the problem of choosing a kernel model into one of finding an *optimal* linear combination of the members of the kernel family. Using this approach there is no need to predefine a kernel; instead, a final kernel is constructed according to the specific classification problem to be solved. In this paper we also use a mixture model representation[3]; however, our approach does not depend on and is not attached to any specific machine learning algorithm (for classification, regression, inference, etc). Instead, inspired by the fact that most kernels can be seen as similarity functions, we rely on explicit conditions on similarity values among subset of points in the original training dataset; this is, our formulation allows the user to explicitly ask for conditions that have to be satisfied or reflected in the desired kernel function to be learned.

1.2 Relative Similarity Constraints

As explained above, we concentrate on examples of proximity comparisons among triples of objects of the type *object i is more like object j than object k*. In choosing this type of relative relationships, we were inspired primarily by [Athitsos *et al.*, 2004; Cohen *et al.*, 1998; Schultz and Joachims, 2003; Rosales and Fung, 2006], where these relationships are defined with respect to distances or rankings. The use of relative similarity constraints of this form for kernel design offers different challenges. Note that we are not interested in preserving absolute similarities, which are in general much more difficult to obtain (absolute similarities imply relative similarities, but the converse is not true).

One important observation for kernel design is that the kernel conditions implied by these relationships are linear relationships among individual kernel entries in the *kernel matrix* (defined over the points of interest) and can be expressed as linear constraints. In addition to this, unlike any of the methods referred to above, by restricting the kernel family to have only kernels that depend in one original feature at the time, we are able to learn kernels that depend on a minimal set of the original features. This can be seen as implicitly performing feature selection with respect to the original data space. In combination with the benefits of a new linear programming approach, the formulation proposed in this paper has a unique set of attributes with significant advantages over recent approaches for kernel design.

1.3 Notation and background

In the following, vectors will be assumed to be column vectors unless transposed to a row vector by a superscript $\top$. The scalar (inner) product of two vectors $\mathbf{x}$ and $\mathbf{y}$ in the d-dimensional real space $\Re^d$ will be denoted by $\mathbf{x}^\top \mathbf{y}$. The 2-norm and 1-norm of $\mathbf{x}$ will be denoted by $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_1$ respectively. A column vector of ones of arbitrary dimension will be denoted by $\vec{e}$, and one of zeros will be denoted by $\vec{0}$. A kernel matrix for a given set of points will be denoted by $\mathbf{K}$, individual components by K_{ij}, and kernel functions by $K()$ or simply K.

2 A Linear programming formulation for kernel design

2.1 Using a linear combination of Kernels

Let us say we are given a set of points $\mathbf{x}_k \in \Re^D$ with $k = \{1, ..., N\}$ for which an appropriate similarity function is unknown or expensive to compute. In addition we are given information about a few relative similarity comparisons. Formally, we are given a set $\mathcal{T} = \{(i,j,k) | K(\mathbf{x}_i, \mathbf{x}_j) > K(\mathbf{x}_i, \mathbf{x}_k)\}$ for some *kernel* function K. The kernel K is not known explicitly, instead a user may only be able to provide these similarity relationships sparsely or by example. We are interested in finding a kernel function $K()$ that satisfies these relationships.

For the rest of the paper, let us suppose that instead of the kernel K being defined by a single kernel mapping (*e.g.,* Gaussian, polynomial, etc.), the kernel K is instead composed of a linear combination of kernel functions $K_j, j = \{1, \ldots, k\}$, as below:

$$K^\alpha(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{k} \alpha_j K_j(\mathbf{x}, \mathbf{y}). \qquad (1)$$

[3]But note that the elements of the mixture could be functions that are not kernels.

As pointed out in [Lanckriet *et al.*, 2004], the set $\Omega = \{K_1(\mathbf{x},\mathbf{y}),\ldots,K_k(\mathbf{x},\mathbf{y})\}$ can be seen as a predefined set of initial *guesses* of the kernel matrix. Note that the set Ω could contain very different kernel matrix models, all with different parameter values. The goal is then to find a kernel function K^α that satisfies relative kernel value constraints (specified by the user as a set $\mathcal{T}$). A general formulation for achieving this is given by:

$$\begin{array}{rrcl} \min_{\alpha,\epsilon} & \sum_t \epsilon_t + \gamma h(\mathbf{K}^\alpha) & & \\ \text{s.t.} & & & \\ \forall(i,j,k)\in\mathcal{T} & K^\alpha(\mathbf{x}_i,\mathbf{x}_j)+\epsilon_t & > & K^\alpha(\mathbf{x}_i,\mathbf{x}_k) \\ \forall t & \epsilon_t & \geq & 0 \\ & \mathbf{K}^\alpha & \succ & 0, \end{array} \quad (2)$$

where ϵ_t are slacks variables, t indexes $\mathcal{T}$, the function $h(\mathbf{K}^\alpha)$ is a regularizer on the kernel matrix $\mathbf{K}^\alpha$ (or alternatively the kernel function K^α) for capacity control, and the parameter $\gamma \geq 0$ controls the trade-off between constraint satisfaction and regularization strength (usually obtained by tuning). Note that this formulation seems sufficient, we can optimize the set of values α_i in order to obtain a positive semidefinite (PSD) linear combination $K^\alpha(\mathbf{x},\mathbf{y}) = \sum_{j=1}^k \alpha_j K_j(\mathbf{x},\mathbf{y})$ suitable for the specific task at hand. This formulation, however requires solving a relatively expensive semidefinite program (SDP). Enforcing $\alpha_i \geq 0, \forall i$, [Fung *et al.*, 2004; Lanckriet *et al.*, 2004] and moreover restricting all the kernels in the set Ω to be PSD results in $\mathbf{K}^\alpha$ being PSD. However this strongly limits the space of attainable kernel functions K^α.

Instead of using these restrictions, we explore an alternative, general definition that allows us to consider any kernel function in Ω and any $\alpha_k \in \Re$ without compromising computational efficiency (*e.g.,* without requiring us to solve a SDP). For this we will explicitly focus on a well characterized subfamily of the PSD matrices: the set of diagonal dominant matrices. In order to provide a better understanding of the motivation for our next formulation we present the following theorem as stated in [Golub and Van Loan, 1996]:

Theorem 1 Diagonal Dominance Theorem *Suppose that $M \in \Re^{D\times D}$ is symmetric and that for each $i = 1,\ldots,n$, we have:*

$$M_{ii} \geq \sum_{j\neq i} |M_{ij}|\,,$$

then M is positive semi-definite (PSD). Furthermore, if the inequalities above are all strict, then M is positive definite.

Based on the diagonal dominance theorem (for matrices with positive diagonal elements) above, by simplifying the notation letting $K_{ij}^\alpha \equiv K^\alpha(\mathbf{x}_i,\mathbf{x}_j)$, we arrive to the following alternative formulation:

$$\begin{array}{rrcl} \min_{\alpha,\epsilon} & \sum_{t=1}^T \epsilon_t + \gamma\,\|\alpha\|_1 & & \\ \text{s.t} & & & \\ \forall t=(i,j,k)\in\mathcal{T} & K_{ij}^\alpha - K_{ik}^\alpha + \epsilon_t & \geq & 1 \\ \forall i\in\mathcal{T} & K_{ii}^\alpha & \geq & \sum_{j\in\mathcal{S},j\neq i}|K_{ij}^\alpha| \\ \forall t & \epsilon_t & \geq & 0, \end{array} \quad (3)$$

where $K_{ij}^\alpha = \sum_{K_i\in\Omega}\alpha_i K_i(\mathbf{x}_i,\mathbf{x}_j)$ and $\mathcal{S}$ is a set of training points (*e.g.,* those points used to define the triples in $\mathcal{T}$). Finally, defining auxiliary variables $r_{ij} \in \Re^+$, $\gamma_1 \geq 0$, and $\gamma_2 \geq 0$, formulation (3) can be rewritten as a linear programming problem in the following way:

$$\begin{array}{rrcl} \min\limits_{\alpha,\epsilon,s,r} & \sum_{t=1}^T \epsilon_t + \gamma_1\sum_i s_i & + & \gamma_2\sum_i K_{ii}^\alpha \\ \text{s.t} & & & \\ \forall t=(i,j,k)\in\mathcal{T} & K_{ij}^\alpha - K_{ik}^\alpha + \epsilon_t & \geq & 1 \\ \forall i\neq j|i,j\in\mathcal{S} & -r_{ij}\leq K_{ij}^\alpha & \leq & r_{ij} \\ \forall i,j\in\mathcal{S} & K_{ii}^\alpha - \sum_{j\neq i} r_{ij} & \geq & 0 \\ \forall j\in\{1,...,k\} & -s_j\leq\alpha_j & \leq & s_j \\ \forall t & \epsilon_t & \geq & 0, \end{array} \quad (4)$$

where $s \in \Re^k$ (same dimensionality as α) and $s_j \geq 0$. Note that to simplify notation we have overloaded the variables i, j, k (but their meaning should be clear from the context).

In order to better understand the motivation for formulation (4) it is important to note that:

(*i*) Minimizing $\sum_i K_{ii}^\alpha$ is equivalent to minimizing $\sum_{\substack{j\\ j\neq i}} r_{ij}$ since $K_{ii}^\alpha \geq \sum_{\substack{j=1\\ j\neq i}}^k r_{ij}, \quad \forall i$.

(*ii*) Since we are implicitly minimizing $\sum_{\substack{j\\ j\neq i}} r_{ij}$, at the optimal solution $\{\alpha^*, r^*, \epsilon^*, s^*\}$ to problem (4), we have that:

$$0 \geq r_{ij}^* = \left|K_{ij}^{\alpha^*}\right|, \forall(i,j\in\mathcal{S}, i\neq j)$$

(*iii*) Combining (*i*) and (*ii*) we obtain:

$$\forall(i) K_{ii}^{\alpha^*} \geq \sum_j r_{ij}^* = \sum_j |K_{ij}^{\alpha^*}|$$

which implies that K^{α^*} is diagonal dominant and hence positive semidefinite.

2.2 Learning kernels that depend on fewer input features ($\tilde{K}$)

Next, we will modify formulation (4) in order to learn kernels that depend in a small subset of the original input features. As far as we know, all of the existing direct objective optimization methods for feature selection with kernels perform feature selection in the implicit feature space induced by the kernel and not with respect to the original feature space. Doing this using traditional kernels, leads to nonlinear, nonconvex optimization problems that are difficult to solve due to the nonlinearity relations among the original features introduced by the kernel mappings.

In order to overcome this difficulty we propose a simple but effective idea. Instead of using kernels that depend on all the features we will only consider a set $\bar{\Omega}$ comprised of *weak* kernels that depend on only one feature at a time. For example, if $\mathbf{x}_i = (x_i^1,\ldots,x_i^f,\ldots,x_i^D)$ and $\mathbf{x}_j = (x_j^1,\ldots,x_j^f,\ldots,x_j^D)$ are two vectors on $\Re^D$, then a weak Gaussian kernel only depending on feature f is defined by:

$$K(f)(\mathbf{x}_i,\mathbf{y}_j) = \exp(-\mu\left\|x_i^f - x_j^f\right\|_2^2) \quad (5)$$

Let us denote by I_f the set of indices i of kernels $K_i \in \bar{\Omega}$ that only depend on feature f for $f \in \{1, \ldots, D\}$. Then, any linear combination of weak kernels in $\bar{\Omega}$ can be written as:

$$K_{ij}^{\alpha} = K^{\alpha}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{f=1}^{D} \sum_{p \in I_f} \alpha_p K_p(\mathbf{x}_i, \mathbf{x}_j) \tag{6}$$

Note that if $\alpha_p = 0,\ \forall p \in I_f$ for a given f, this implies that K_{ij}^{α} does not depend on the original feature f. This motivates our next formulation for feature selection that uses *weak one-dimensional* kernels:

$$\begin{array}{rrcl}
\min\limits_{\alpha,\epsilon,s} & \sum_{t=1}^{T} \epsilon_t + \gamma_1 \sum_f s_f & + & \gamma_2 \sum_i K_{ii}^{\alpha} \\
\text{s.t} & & & \\
\forall t = (i,j,k) \in \mathcal{T} & K_{ij}^{\alpha} - K_{ik}^{\alpha} + \epsilon_t & \geq & 1 \\
\forall i \neq j & -r_{ij} \leq K_{ij}^{\alpha} & \leq & r_{ij} \\
\forall i & K_{ii}^{\alpha} - \sum_j r_{ij} & \geq & 0 \\
\forall f, \forall p \in I_f & -s_f \leq \alpha_p & \leq & s_f \\
 & \epsilon_t & \geq & 0,
\end{array} \tag{7}$$

where now s is indexed by the feature number f rather than the kernel number alone. It is interesting to note that for each feature f, formulation (7) is minimizing $M_f = \max\{|\alpha_p| / p \in I_f\}$. This is appropriate since:

$$\begin{array}{rcl}
M_f = 0 & \Rightarrow & |\alpha_p| \leq 0, \forall p \in I_f \\
 & \Rightarrow & \alpha_p = 0, \forall p \in I_f \\
 & \Rightarrow & \forall i, j, K_{ij}^{\alpha} \text{ does not depend on feature} f.
\end{array} \tag{8}$$

3 Numerical Evaluation

For our experimental evaluation we used a collection of nine publicly available datasets, part of the UCI repository[4]. A summary of these datasets is shown in Table 1. These datasets are commonly used in machine learning as a benchmark for performance evaluation. Our choice of datasets is motivated primarily by their use in evaluating a competing approach [Xing *et al.*, 2002] aimed to learn distance functions.

We evaluate our approach against single kernels by comparing against the standard Gaussian (at various widths), linear, and polynomial kernels. These kernels are also the ones used as the basis for our mixture kernel design; thus it is a reasonable baseline comparison. We compare both of our formulations. One formulation attempts to perform implicit feature selection by defining weak kernels, while the other uses full kernel matrices (that depend on all input dimensions).

We have also chosen to compare our formulation against that proposed in [Xing *et al.*, 2002]. This obeys various reasons. In addition to being a state-of-the-art method, the primary reason for our choice is that it uses a similar (but not identical) type of supervision, as explained below. Unlike other related approaches, computer code and data has been made public for this method[5]. In addition, this method outperformed a constrained version of K-means [Wagstaff *et al.*, 2001] in the task of finding *good* clusterings.

[4]http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[5]Data for all experiments and code for [Xing *et al.*, 2002] was downloaded from http://www.cs.cmu.edu/∼epxing/papers/. The class for dataset 1 was obtained by thresholding the median value attribute to 25K.

Table 1: Benchmark Datasets

Name	Pts (N)	Dims (D)	Classes
1 Housing-Boston	506	13	2
2 Ionosphere	351	34	2
3 Iris	150	4	3
4 Wine	178	13	3
5 Balance Scale	625	4	3
6 Breast-Cancer Wisc.	569	30	2
7 Soybean Small	47	35	4
8 Protein	116	20	6
9 Pima Diabetes	768	8	2

3.1 Evaluation Settings

The datasets employed in these experiments are generally used for classification since class labels are available. However, the various methods to be compared here do not require explicit class labels. The method introduced in this paper requires relative similarity information among a subset of points (clearly class labels provide more information). We use the available class labels to generate a set of triples with similarity comparisons that respect the classes. More explicitly, given a randomly chosen set of three points (from the training set), if two of these belong to the same class and a third belongs to a different class, then we place this triple in our set $\mathcal{T}$ (*i.e.*, i and j are the points in the same class, k is the remaining point). For the case of [Xing *et al.*, 2002], the supervision is in the form of two sets, one called a *similar* set and the other a *dissimilar* set. In order to identify these sets, we can again use the class labels, now to build a similar set of pairs (likewise for a dissimilar set of pairs). Given this level of supervision, this method attempts to find an optimal Mahalanobis distance matrix to have same-class points closer to each other than different-class points (see [Xing *et al.*, 2002] for details).

For every triple $(i, j, k) \in \mathcal{T}$ used in our approach for learning, we use $(i, j) \in \mathcal{S}$ and $(i, k) \in \mathcal{D}$ for learning in [Xing *et al.*, 2002]; where $\mathcal{S}$ and $\mathcal{D}$ are the similar and dissimilar sets. We believe this provides a fair level of supervision for both algorithms since roughly the same information is provided. It is possible to obtain a superset of $\mathcal{T}$ from $\mathcal{S}$ and $\mathcal{D}$, and by construction $\mathcal{S}$ and $\mathcal{D}$ can be obtained from $\mathcal{T}$.

In order to evaluate performance for the various methods, we use a $0.85/0.15$ split of the data into training and testing (for the methods where training is required). From the training portion, we generate 1500 triples, as explained above, for actual training. This information is provided, in the appropriate representation, to both algorithms. For testing, we repeatedly choose three points at random, and if their class labels imply that any two points are more similar to each other than to a third (*i.e.*, again if two points have the same class and a third has a different class label), then we check that the correct relationships were learned. That is, whether the two points in the same class are more similar (or closer) to each other than any of these points (chosen at random) to the third point. This same measure is used for all algorithms. Thus, we define the *percentage correct* simply as the proportion of points from the test set (sampled at random) that respect the class-implied similarity or distance relationship.

Our method requires setting two balancing parameters. We set them by using cross validation by splitting the training set in two halves. The values tested (for both parameters) were $\{10^{-4}, 10^{-2}, 10^{-1}, 1, 10, 10^2\}$. These parameters have an effect on the number of dimensions employed since a higher value for γ_1 favors using fewer kernels (or dimensions). Likewise, larger values for γ_2 favors kernel matrices with a smaller trace.

3.2 Discussion and Results

Fig. 1 shows the performance of our approach compared against various Gaussian and polynomial kernels. A linear kernel performed almost identically to the polynomial kernel of degree two and it is omitted in the graph. The mean and standard deviation (of the performance measure) for each individual kernel was computed from 10 random samplings of the dataset. In order to be consistent across approaches, the number of samples used for testing was set to 1000 triples.

One could expect an optimal mixture of kernels to provide a higher performance in both cases (full and weak kernel mixtures) when compared with single kernels. This is generally the case. It can be seen in the figure that in most cases single predetermined kernels are suboptimal. In the case of weak kernels, the performance is always better than single kernels. However, for the case of mixtures of full kernels there are cases where a single kernel provides a higher performance in average. This can be at least due to two reasons: (1) simply unlucky selection (sampling) of test points. In datasets $1, 3$, and 9, the standard deviation is large enough to justify this possibility. A much more interesting reason is (2) the imposed restrictions on the solution space of the mixture kernel (*e.g.,* dataset 6). Recall that we concentrated our solution on a subspace of the PSD matrices, that of diagonal dominant matrices. If the full cone of PSD matrices were to be incorporated as solution space, the kernel mixture could perform as good as any base kernel (since a mixture of a single kernel is a valid solution). Note however that, although this is possible, one would be required to pay much higher computational costs (*e.g.,* for large datasets or number of constraints).

Interestingly, the performance for the mixture of weak kernels is superior. This can be due to the larger number of degrees of freedom (the number of α mixing parameters is larger) or also a feature selection effect on overfitting. However, even though this is the case for the datasets considered in this paper (and results suggest that this may often be the case), we remark that this result is not theoretically guaranteed since non-linear interactions among multiple dimensions or features may not be representable using a linear combination of single features.

Fig. 2 shows the average optimal number of dimensions found by this process in a 10-fold cross validation experiment and the corresponding one-standard-deviation error bars. The number of dimensions was identified by counting the number of α's larger than 0.01. This automatic choice of dimensionality was done using cross-validation as explained above and is a valuable property of the method presented. Note that the method effectively reduces the number of features for all but one dataset.

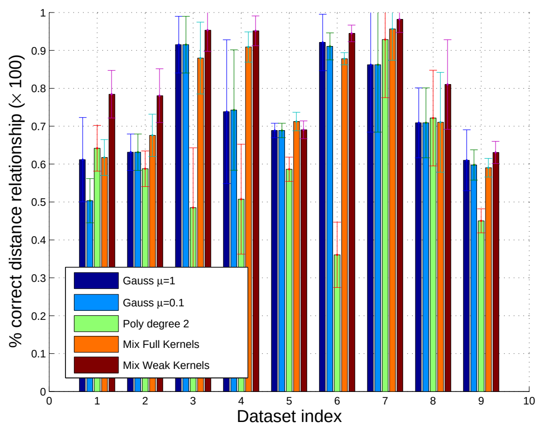

Figure 1: Performance of single kernels and our approachs in the nine UCI datasets. We show both instances of our approach (using full kernel matrices and weak kernels). Bars show performance results on 10 random splits of training/test points (training only applicable to kernel mixtures). Performance is measured in terms of the percentage of randomly chosen points (1000) from test set whose distance relationship respect the class labels. The number of triples used for training for all runs was 1500. Error bars show one standard deviation.

Fig. 3 shows the comparison of the present formulation (using weak kernels) against [Xing *et al.*, 2002]. We show percentage correct averaged over 10 random splits of the data along with one-standard-deviation bars. For each of the 10 splits, 1000 triples from the test set are randomly chosen. When comparing the performance of both methods, we note that, except for dataset 5, our method clearly outperforms the competing approach. Interestingly, this dataset was the same for which the optimal number of dimensions was determined to always be equal to the original dimensionality.

In addition to the implicit non-linear representations implied by the kernels employed, we believe that a key reason for the superior performance of the mixture of weak kernels is the automatic identification of relevant dimensions. This reduction in dimensionality appears to provide an important advantage at the time of generalization. It is generally accepted that a simpler representation is preferable (*e.g.,* [Blumer *et al.*, 1987]) and it can reduce overfitting in practice.

From a computational efficiency perspective at test time, being able to represent the original data more succinctly is especially advantageous. In particular, when similarities can be calculated directly using a low-dimensional representation, computational time savings can be significant for on-line applications. The projection step in this approach can be precomputed off-line. In retrieval applications (*e.g.,* query-by-example), the objects can be stored in their low-dimensional representation. From a conceptual point of view, this formulation also has the advantage of providing a more effective tool for understanding the data since it can identify whether variables (dimensions) are of high or low relevance for a task of interest.

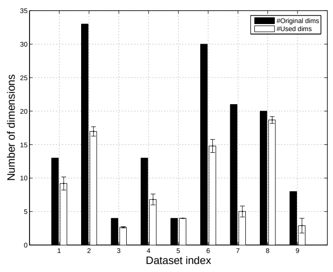

Figure 2: Dimensionality reduction. Total number of dimensions and average number of dimensions (along with one-standard-deviation error bars) found by our algorithm for each dataset using the optimal parameters γ_1, γ_2. Averages are computed over 10 random splits of training/test points, and 1500 triples per run.

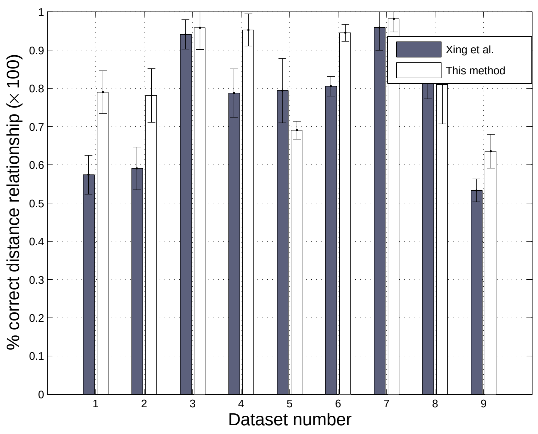

Figure 3: Performance comparison between competing approach and our weak kernel approach in the nine UCI datasets. Bars show performance results on 10 random splits of training/test points. Performance is measured in terms of the percentage of randomly chosen points (1000) from test set whose distance relationship respect the class labels. The number of triples used for training for all runs was 1500. Error bars show one standard deviation.

4 Conclusions and Future Work

We presented a novel formulation for learning kernel functions from relative similarity information. Our general formulation consisted of a convex optimization problem requiring semidefinite programming to be solved. We designed a practical approximate problem, requiring linear programming instead. In particular, we showed how the diagonal dominance constraint on the kernel matrix leads to a general problem that can be solved very efficiently. In addition to this, we have shown how to extend the formulation to allow for implicit feature selection using a combination of kernels. Experiments indicated that our weak kernel formulation outperforms a state-of-the-art approach that uses similar information to learn a distance (rather than a similarity function).

Although relative similarity constraints are used in this paper, other constraints could potentially be imposed. For example, we could have sign or magnitude restrictions on certain kernel components, we can impose local isometry constraints (similar to [Weinberger *et al.*, 2004]), or we can request the kernel to preserve local distances.

We believe this paper has potential implications in other problems. Our results suggest that the SDP problem could, in some cases, be replaced by a linear programming problem. We suggested one way to achieve this goal; this has the potential to be applied to other problems whose solution involves SDP.

References

[Athitsos *et al.*, 2004] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Boostmap: A method for efficient approximate similarity rankings. In *Computer Vision and Pattern Recognition*, 2004.

[Bennett *et al.*, 2002] K. Bennett, M. Momma, and M. Embrechts. Mark: a boosting algorithm for heterogeneous kernel models. In *Proceedings Knowledge Discovery and Data Mining*, 2002.

[Blumer *et al.*, 1987] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth. Occam's razor. *Information Processing Letters*, 24:377–380, 1987.

[Cohen *et al.*, 1998] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. In *Advances in Neural Information Processing Systems 10*, 1998.

[Cristianini and Shawe-Taylor, 2000] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[Fung *et al.*, 2004] G. Fung, M. Dundar, J. Bi, and B. Rao. A fast iterative algorithm for fisher discriminant using heterogeneous kernels. In *Proceedings of the twenty-first international conference on Machine learning*. ACM Press, 2004.

[Golub and Van Loan, 1996] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, Maryland, 3rd edition, 1996.

[Graepel, 2002] T. Graepel. Kernel matrix completion by semidefinite programming. In *Proceedings of the International Conference on Neural Networks, ICANN*, 2002.

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

[Rosales and Fung, 2006] R. Rosales and G. Fung. Learning sparse metrics via linear programming. In *Proceedings Knowledge Discovery and Data Mining*, 2006.

[Schultz and Joachims, 2003] M. Schultz and T. Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems*, 2003.

[Wagstaff *et al.*, 2001] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *International Conference on Machine Learning*, 2001.

[Weinberger *et al.*, 2004] K. Weinberger, F. Sha, and L. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the Twenty First International Confernence on Machine Learning*, 2004.

[Xing *et al.*, 2002] E.P. Xing, A.Y. Ng, M.I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side information. In *Advances in Neural Information Processing Systems*, 2002.