Distance Metric Learning for Kernel Machines

Zhixiang (Eddie) Xu xuzx@cse.wustl.edu
Department of Computer Science and Engineering
Washington University in St. Louis
Saint Louis, MO 63130, USA

Kilian Q. Weinberger kilian@wustl.edu
Department of Computer Science and Engineering
Washington University in St. Louis
Saint Louis, MO 63130, USA

Olivier Chapelle Olivier@chapelle.cc
Criteo
Palo Alto, CA 94301

Editor:

Abstract

Recent work in metric learning has significantly improved the state-of-the-art in k-nearest neighbor classification. Support vector machines (SVM), particularly with RBF kernels, are amongst the most popular classification algorithms that uses distance metrics to compare examples. This paper provides an empirical analysis of the efficacy of three of the most popular Mahalanobis metric learning algorithms as pre-processing for SVM training. We show that none of these algorithms generate metrics that lead to particularly satisfying improvements for SVM-RBF classification. As a remedy we introduce support vector metric learning (SVML), a novel algorithm that seamlessly combines the learning of a Mahalanobis metric with the training of the RBF-SVM parameters. We demonstrate the capabilities of SVML on nine benchmark data sets of varying sizes and difficulties. In our study, SVML outperforms all alternative state-of-the-art metric learning algorithms in terms of accuracy and establishes itself as a serious alternative to the standard Euclidean metric with model selection by cross validation.

Keywords: metric learning, distance learning, support vector machines, semi-definite programming, Mahalanobis distance

1. Introduction

Many machine learning algorithms, such as k-nearest neighbors (kNN) (Cover and Hart, 1967), k-means (Lloid, 1982) or support vector machines (SVM) (Cortes and Vapnik, 1995) with shift-invariant kernels, require a distance metric to compare instances. These algorithms rely on the assumption that semantically similar inputs are close, whereas semantically dissimilar inputs are far away. Traditionally, the most commonly used distance metrics are uninformed norms, like the Euclidean distance. In many cases, such uninformed norms are sub-optimal. To illustrate this point, imagine a scenario where two researchers want to classify the same data set of facial images. The first one classifies people by age, the second by gender. Clearly, two images that are similar according to the first researcher's setting might be dissimilar according to the second's.

Uninformed norms ignore two important contextual components of most machine learning applications. First, in supervised learning the data is accompanied by labels which essentially encode the semantic definition of similarity. Second, the user knows which machine learning algorithm will be used. Ideally, the distance metric should be tailored to the particular setting at hand, incorporating both of these considerations.

A generalization of the Euclidean distance is the Mahalanobis distance (Mahalanobis, 1936). Recent years have witnessed a surge of innovation on Mahalanobis pseudo-metric learning (Davis et al., 2007; Globerson and Roweis, 2005; Goldberger et al., 2005; Shental et al., 2002; Weinberger et al., 2006). Although these algorithms use different methodologies, the common theme is moving similar inputs closer and dissimilar inputs further away — where similarity is generally defined through class membership. This transformation can be learned through convex optimization with pairwise constraints (Davis et al., 2007; Weinberger et al., 2006), gradient descent with soft neighborhood assignments (Goldberger et al., 2005), or spectral methods based on second-order statistics (Shental et al., 2002).

Typically, the Mahalanobis metric learning algorithms are used in a two-step approach. First the metric is learned, then it is used for training the classifier or clustering algorithm of choice. The resulting distances are semantically more meaningful than the plain Euclidean distance as they reflect the label information. This makes them particularly suited for the k-nearest neighbor rule, leading to large improvements in classification error (Davis et al., 2007; Globerson and Roweis, 2005; Goldberger et al., 2005; Shental et al., 2002; Weinberger et al., 2006). In fact, several algorithms explicitly mimic the k-NN rule and minimize a surrogate loss function of the corresponding leave-one-out classification error on the training set (Goldberger et al., 2005; Weinberger et al., 2006).

Although the k-nearest neighbor rule can be a powerful classifier especially in settings with many classes, it comes with certain limitations. For example, the entire training data needs to be stored and processed during test time. Also, in settings with fewer classes (especially binary) it is generally outperformed by Support Vector Machines (Cortes and Vapnik, 1995). Because of their high reliability as out-of-the-box classifiers, SVMs have become one of the quintessential classification algorithms in many areas of science and beyond. An important part of using SVMs is the right choice of kernel. The kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$ encodes the similarity between two input vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. There are many possible choices for such a kernel function. One of the most commonly used kernels is the Radial Basis Function (RBF) kernel (Schölkopf and Smola, 2002), which itself relies on a distance metric.

This paper considers metric learning for support vector machines. As a first contribution, we review and investigate several recently published kNN metric learning algorithms for the use of SVMs with RBF kernels. We demonstrate empirically that these approaches do not reliably improve SVM classification results up to statistical significance. As a second contribution, we derive a novel metric learning algorithm that specifically incorporates the SVM loss function during training. Here, we learn the metric to minimize the validation error of the SVM prediction at the same time that we train the SVM. This is in contrast to the two-step approach of first learning a metric and then training the SVM classifier with the resulting kernel. This algorithm, which we refer to as Support Vector Metric

Learning (SVML), is particularly useful for three reasons. First, it achieves state-of-the-art classification results and clearly outperforms other metric learning algorithms that are not explicitly geared towards SVM classification. Second, it provides researchers outside of the machine-learning community a convenient way to automatically pre-process their data before applying SVMs.

This paper is organized as follows. In Section 2, we introduce necessary notation and review some background on SVMs. In Section 3 we introduce several recently published metric learning algorithms and report results for SVM-RBF classification. In Section 4 we derive the SVML algorithm and some interesting variations. In Section 5, we evaluate SVML on nine publicly available data sets featuring a multitude of different data types and learning tasks. We discuss related work in Section 6 and conclude in Section 7.

2. Support Vector Machines

Let the training data consist of input vectors $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\} \in \mathcal{R}^d$ with corresponding discrete class labels $\{y_1, \ldots, y_n\} \in \{+1, -1\}$. Although our framework can easily be applied in a multi-class setting, for the sake of simplicity we focus on binary scenarios, restricting y_i to two classes.

There are several reasons why SVMs are particularly popular classifiers. First, they are linear classifiers that involve a quadratic minimization problem, which is convex and guarantees perfect reproducibility. Furthermore, the maximum margin philosophy leads to reliably good generalization error rates (Vapnik, 1998). But perhaps most importantly, the *kernel-trick* (Schölkopf and Smola, 2002) allows SVMs to generate highly non-linear decision boundaries with low computational overhead. More explicitly, the kernel-trick maps the input vectors $\mathbf{x}_i$ implicitly into a higher (possibly infinite) dimensional feature space with a non-linear transformation $\phi : \mathcal{R}^d \to \mathcal{H}$. Training a linear classifier directly in this high dimensional feature space $\mathcal{H}$ would be computationally infeasible if the vectors $\phi(\mathbf{x}_i)$ were accessed explicitly. However, SVMs can be trained completely in terms of inner-products between input vectors. With careful selection of $\phi()$, the inner-product $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ can be computed efficiently even if computation of the mapping $\phi()$ itself is infeasible. Let the kernel function be $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ and the $n \times n$ kernel matrix be $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. The optimization problem of SVM training can be expressed entirely in terms of the kernel matrix $\mathbf{K}$. For the sake of brevity, we omit the derivation and refer the interested reader to one of many detailed descriptions thereof (Schölkopf and Smola, 2002). The resulting classification rule of a test point $\mathbf{x}_t$ becomes

$$h(\mathbf{x}_t) = \operatorname{sign}(\sum_{j=1}^{n} \alpha_j y_j k(\mathbf{x}_j, \mathbf{x}_t) + b), \tag{1}$$

where b is the offset of the separating hyperplane and $\alpha_1, \ldots, \alpha_n$ are the dual variables corresponding to the inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$. In the case of the hard-margin SVM, the parameters α_i are learned with the following quadratic optimization problem

$$\min_{\alpha_1, \ldots, \alpha_n} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

$$\text{subject to : } \sum_{i=1}^{n} \alpha_i y_i = 0 \text{ and } \alpha_i \geq 0. \tag{2}$$

The optimization problem (2) ensures that all inputs $\mathbf{x}_i$ with label $y_i = -1$ are on one side of the hyperplane, and those with label $y_j = +1$ are on the other. These hard constraints might not always be feasible, or in the interest of minimizing the generalization error (*e.g.* in the case of noisy data). Relaxing the constraints can be performed simply by altering the kernel matrix to

$$\mathbf{K} \leftarrow \mathbf{K} + \frac{1}{C}\mathbf{I}^{n \times n}. \tag{3}$$

Solving (2) with a kernel matrix (3) is equivalent to a squared-penalty of the violations of the separating hyperplane (Cortes and Vapnik, 1995). This formulation requires no explicit slack variables in the optimization problem and therefore simplifies the derivations of the following sections.

2.1 RBF Kernel

There are many different kernel functions that are suitable for SVMs. In fact, any function $k(\cdot, \cdot)$ is a well-defined kernel as long as it is positive semi-definite (Schölkopf and Smola, 2002). The Radial Basis Function (RBF)-Kernel is defined as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{-d^2(\mathbf{x}_i, \mathbf{x}_j)}, \tag{4}$$

where $d(\cdot, \cdot)$ is a dissimilarity measure that must ensure positive semidefiniteness of $k(\cdot, \cdot)$. The most common choice is the re-scaled squared Euclidean distance, defined as

$$d^2(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{\sigma^2}(\mathbf{x}_i - \mathbf{x}_j)^\top (\mathbf{x}_i - \mathbf{x}_j), \tag{5}$$

with *kernel width* $\sigma > 0$. The RBF-kernel is one of the most popular kernels and yields reliable good classification results. Also, with careful selection of C, SVMs with RBF-kernels have been shown to be consistent classifiers (Steinwart, 2002).

2.2 Relationship with kNN

The k-nearest neighbor classification rule predicts the label of a test point $\mathbf{x}_t$ through a majority vote amongst its k nearest neighbors. Let $\eta_j(\mathbf{x}_t) \in \{0, 1\}$ be the neighborhood indicator function of a test point $\mathbf{x}_t$, where $\eta_j(\mathbf{x}_t) = 1$ if and only if $\mathbf{x}_j$ is one of the k nearest neighbors of $\mathbf{x}_t$. The kNN classification rule can then be expressed as

$$h(\mathbf{x}_t) = \text{sign}(\sum_{j=1}^{n} \eta_j(\mathbf{x}_t) y_j). \tag{6}$$

Superficially, the classification rule in (6) very much resembles (1). In fact, one can interpret the SVM-RBF classification rule in (1) as a *"soft"*-nearest neighbor rule. Instead of the zero-one step function $\eta_j(\mathbf{x}_t)$, the training points are weighted by $\alpha_j k(\mathbf{x}_t, \mathbf{x}_j)$. The classification is still local-neighborhood based, as $k(\mathbf{x}_t, \mathbf{x}_j)$ decreases exponentially with increasing distance $d(\mathbf{x}_t, \mathbf{x}_j)$. The SVM optimization in (2) assigns appropriate weights $\alpha_j \geq 0$ to ensure that, on the leave-one-out training set, the majority vote is correct for all data points by a large margin.

3. Metric Learning

It is natural to ask if the SVM classification rule can be improved with better adjusted metrics than the Euclidean distance. A commonly used generalization of the Euclidean metric is the Mahalanobis metric (Mahalanobis, 1936), defined as

$$d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{M}(\mathbf{x}_i - \mathbf{x}_j)}, \tag{7}$$

for some matrix $\mathbf{M} \in \mathcal{R}^{d\times d}$. The matrix $\mathbf{M}$ must be semi-positive definite ($\mathbf{M} \succeq 0$), which is equivalent to requiring that it can be decomposed into $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, for some matrix $\mathbf{L} \in \mathcal{R}^{r\times d}$. If $\mathbf{M} = \mathbf{I}^{d\times d}$, where $\mathbf{I}^{d\times d}$ refers to the identity matrix in $\mathcal{R}^{d\times d}$, (7) reduces to the Euclidean metric. Otherwise, it is equivalent to the Euclidean distance after the transformation $\mathbf{x}_i \to \mathbf{L}\mathbf{x}_i$. Technically, if $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$ is a singular matrix, the corresponding Mahalanobis distance is a *pseudo-metric*[1]. Because the distinction between pseudo-metric and metric is unimportant for this work, we refer to both as *metrics.* As the distance in (7) can equally be parameterized by $\mathbf{L}$ and $\mathbf{M}$ we use $d_{\mathbf{M}}$ and $d_{\mathbf{L}}$ interchangeably.

In the following section, we will introduce several approaches that focus on Mahalanobis metric learning for k-nearest neighbor classification.

3.1 Neighborhood component analysis

Goldberger et al. (2005) propose Neighborhood Component Analysis (NCA), which minimizes the expected leave-one-out classification error under a probabilistic neighborhood assignment. For each data point or query, the neighbors are drawn from a softmax probability distribution. The probability of sampling $\mathbf{x}_j$ as a neighbor of $\mathbf{x}_i$ is given by:

$$p_{ij} = \begin{cases} \frac{e^{-d^2_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_j)}}{\sum_{k\neq i} e^{-d^2_{\mathbf{L}}(\mathbf{x}_i,\mathbf{x}_k)}} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{8}$$

Let us define an indicator variable $y_{ij} \in \{0, 1\}$ where $y_{ij} = 1$ if and only if $y_i = y_j$. With the probability assignment described in (8), we can easily compute the expectation of the leave-one-out classification *accuracy* as

$$A_{loo} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} p_{ij} y_{ij}. \tag{9}$$

NCA uses gradient ascent to maximize (9). The advantage of the probabilistic framework over regular kNN is that (9) is a continuous, differentiable function with respect to the linear transformation $\mathbf{L}$. By contrast, the leave-one-out error of regular kNN is not continuous or differentiable. The two down-sides of NCA are its relatively high computational complexity and non-convexity of the objective.

[1] A pseudo-metric is not require to preserve identity, i.e. $d(\mathbf{x}_i, \mathbf{x}_j) = 0 \iff \mathbf{x}_i = \mathbf{x}_j$.

3.2 Large Margin Nearest Neighbor Classification

Large Margin Nearest Neighbor (LMNN), proposed by Weinberger et al. (2006), also mimics the leave-one-out error of kNN. Unlike NCA, LMNN employs a convex loss function, and encourages local neighborhoods to have the same labels by pushing data points with different labels away and pulling those with similar labels closer. The authors introduce the concept of *target neighbors*. A target neighbor of a training datum $\mathbf{x}_i$ are data points in the training set that *should ideally be* the nearest neighbors (e.g. the closest points under the Euclidean metric with the same class label). LMNN moves these points closer by minimizing

$$\sum_{j \rightsquigarrow i} d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j), \tag{10}$$

where $j \rightsquigarrow i$ indicates that $\mathbf{x}_j$ is a target neighbor of $\mathbf{x}_i$. In addition to the objective (10), LMNN also enforces that no datum with a different label can be closer than a target neighbor. In particular, let $\mathbf{x}_i$ be a training point and $\mathbf{x}_j$ one of its target neighbors. Any point $\mathbf{x}_k$ of *different class membership* than $\mathbf{x}_i$ should be further away than $\mathbf{x}_j$ by a large margin. LMNN encodes this relationship as linear constraints with respect to $\mathbf{M}$.

$$d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k) \geq d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) + 1 \tag{11}$$

LMNN uses semidefinite programming to minimize (10) with respect to (11). To account for the natural limitations of a single linear transformation the authors introduce slack variables. More explicitly, for each triple (i, j, k), where $\mathbf{x}_j$ is a target neighbor of $\mathbf{x}_i$ and $y_k \neq y_i$, they introduce $\xi_{ijk} \geq 0$ which absorbs small violations of the constraint (11). The resulting optimization problem can be formulated as the following semi-definite program (SDP) (Boyd and Vandenberghe, 2004):

$$\boxed{\begin{array}{l} \min\limits_{\mathbf{M} \succeq 0} \sum\limits_{j \rightsquigarrow i} d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) + \mu \sum\limits_{j \rightsquigarrow i, k : y_k \neq y_i} \xi_{ijk} \\ \textbf{subject to}: \\ \quad (1)\ d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_k) - d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \geq 1 - \xi_{ijk} \\ \quad (2)\ \xi_{ijk} \geq 0 \end{array}}$$

Here $\mu \geq 0$ defines the trade-off between minimizing the objective and penalizing constraint violations (by default we set $\mu = 1$).

3.3 Information-Theoretic Metric Learning

Different from NCA and LMNN, Information-Theoretic Metric Learning (ITML), proposed by Davis et al. (2007), does not minimize the leave-one-out error of kNN classification. In contrast, ITML assumes a uni-modal data-distribution and clusters similarly labeled inputs close together while regularizing the learned metric to be close to some pre-defined initial metric in terms of Gaussian cross entropy (for details see Davis et al. (2007)). Similar to LMNN, ITML also incorporates the similarity and dissimilarity as constraints in its optimization. Specifically, ITML enforces that similarly labeled inputs must have a distance smaller than a given upper bound $d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq u$ and dissimilarly labeled points must be further apart than a pre-defined lower bound $d_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \geq l$. If we denote the set of similarly labeled input pairs as S, and dissimilar pairs as D, the optimization problem of ITML is:

$$\boxed{\begin{array}{l} \min\limits_{\mathbf{M} \succeq 0} \operatorname{tr}(\mathbf{M}\mathbf{M}_0^{-1}) - \log\det(\mathbf{M}\mathbf{M}_0^{-1}) \\ \textbf{subject to:} \\ \quad (1)\ d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \leq u \quad \forall (i,j) \in S, \\ \quad (2)\ d^2_{\mathbf{M}}(\mathbf{x}_i, \mathbf{x}_j) \geq l \quad \forall (i,j) \in D. \end{array}}$$

Davis et al. (2007) introduce several variations, including the incorporation of slack-variables. One advantage of the particular formulation of the ITML optimization problem is that the SDP constraint $\mathbf{M} \succeq 0$ does not have to be monitored explicitly through eigenvector decompositions but is enforced implicitly through the objective.

Statistics	**Haber**	**Credit**	**ACredit**	**Trans**	**Diabts**	**Mammo**	**CMC**	**Page**	**Gamma**
#examples	306	653	690	748	768	830	962	5743	19020
#features	3	15	14	4	8	5	9	10	11
#training exam.	245	522	552	599	614	664	770	4594	15216
#testing exam.	61	131	138	150	154	166	192	1149	3804
Metric	**Error Rates**								
Euclidean	27.37	**13.12**	**14.11**	**20.54**	23.46	18.17	26.91	**2.56**	**12.62**
ITML	**26.50**	13.68	14.71	22.86	23.14	18.20	27.67	4.78	21.50
NCA	**26.39**	13.48	**14.10**	22.59	**22.74**	18.17	**26.53**	4.74	N/A
LMNN	**26.70**	13.48	**13.89**	20.81	**22.89**	**17.78**	**26.68**	**2.66**	13.04

Table 1: Error rates of SVM classification with an RBF kernel (all parameters were set by 5-fold cross validation) under various learned metrics.

3.4 Metric Learning for SVM

We evaluate the efficacy of NCA, ITML and LMNN as pre-processing step for SVM classification with an RBF kernel. We used nine data sets from the UCI Machine Learning repository (Frank and Asuncion, 2010) of varying size, dimensionality and task description. The data sets are: Haberman's Survival (Haber), Credit Approval (Credit), Australian Credit Approval (ACredit), Blood Transfusion Service (Trans), Diabetes (Diabts), Mammographic Mass (Mammo), Contraceptive Method Choice (CMC), Page Blocks Classification (Page) and MAGIC Gamma Telescope (Gamma).

For simplicity, we restrict our evaluation to the binary case and convert multi-class problems to binary ones, either by selecting the two most-difficult classes or (if those are not known) by grouping labels into two sets. Table 1 details statistic and classification results on all nine data sets. The best values up to statistical significance (within a 5% confidence interval) are highlighted in bold. To be fair to all algorithms, we re-scale all features to have standard deviation 1. We follow the commonly used heuristic for Euclidean RBF[2] and initialize NCA and ITML with $\mathbf{L}_0 = \frac{1}{d}\mathbf{I}$ for all experiments (where d denotes the

[2] The choice of $\sigma^2 = \#features$ is also the default value for the LibSVM toolbox (Chang and Lin, 2001).

$\#features$). As LMNN is known to be very parameter insensitive, we set μ to the default value of $\mu = 1$. All SVM parameters (C and σ^2) were set by 5-fold cross validation on the training sets, after the metric is learned. The results on the smaller data sets ($n < 1000$) were averaged over 200 runs with random train/test splits, Page Blocks (Page) was averaged over 20 runs and Gamma was run once (here the train/test splits are pre-defined).

In terms of scalability, NCA is by far the slowest algorithm and our implementation did not scale up to the (largest) Gamma data set. LMNN and ITML require comparable computation time (on the order of several minutes for the small- and 1-2 hours for large data sets – for details see Section 6). As a general trend, none of the three metric learning algorithms consistently outperforms the Euclidean distance. Given the additional computation time, it is questionable if either one is a reasonable pre-processing step of SVM-RBF classification. This is in large contrast with the drastic improvements that these metric learning algorithms obtain when used as pre-processing for kNN (Goldberger et al., 2005; Weinberger et al., 2006; Davis et al., 2007). One explanation for this discrepancy could be based on the subtle but important differences between the kNN classification rule (6) and the one of SVMs (1). In the remainder of this paper we will explore the possibility to learn a metric explicitly for the SVM decision rule.

4. Support Vector Metric Learning

As a first step towards learning a metric specifically for SVM classification, we incorporate the squared Mahalanobis distance (7) into the kernel function (4) and define the resulting kernel function and matrix as

$$k_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j) = e^{-(\mathbf{x}_i - \mathbf{x}_j)^\top \mathbf{L}^\top \mathbf{L} (\mathbf{x}_i - \mathbf{x}_j)} \text{ and } \mathbf{K}_{ij} = k_{\mathbf{L}}(\mathbf{x}_i, \mathbf{x}_j). \quad (12)$$

As mentioned before, the typical Euclidean RBF setting is a special case where $\mathbf{L} = \frac{1}{\sigma}\mathbf{I}^{d \times d}$.

4.1 Loss function

In the Euclidean case, a standard way to select the meta parameter σ is through cross-validation. In its simplest form, this involves splitting the training data set into two mutually exclusive subsets: training set T and validation set V. The SVM parameters α_i, b are then trained on T and the outcome is evaluated on the validation data set V. After a gridsearch over several candidate values for σ (and C), the setting that performs the best on the validation data is chosen. For a single meta parameter, search by cross validation is simple and surprisingly effective. If more meta parameters need to be set — in the case of choosing a matrix $\mathbf{L}$, this involves $d \times d$ entries — the number of possible configurations grows exponentially and the gridsearch becomes infeasible.

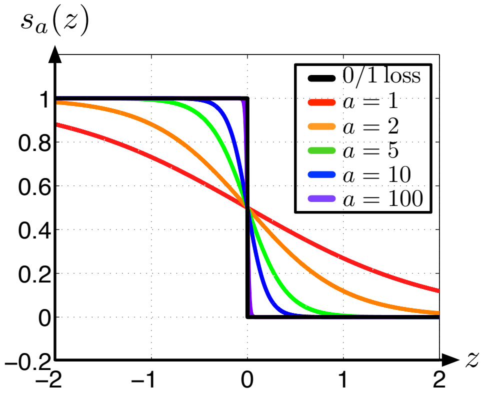

Figure 1: The function $s_a(z)$ is a soft (differentiable) approximation of the zero-one loss. The parameter a adjusts the steepness of the curve.

We follow the intuition of validating meta parameters on a hold-out set of the training data. Ideally, we want to find a metric parameterized by $\mathbf{L}$ that minimize the classification error $\mathcal{E}_V$ on the validation data set

$$\mathbf{L} = \operatorname*{argmin}_{\mathbf{L}} \mathcal{E}_V(\mathbf{L}) \text{ where: } \mathcal{E}_V(\mathbf{L}) = \frac{1}{|V|} \sum_{(\mathbf{x},y)\in V} [h(\mathbf{x}) = y].$$

Here $[h(\mathbf{x}) = y] \in \{0, 1\}$ takes on value 1 if and only if $h(\mathbf{x}) = y$. The classifier $h(\cdot)$, defined in (1) depends on parameters α_i and b, which are re-trained for every intermediate setting of $\mathbf{L}$. Performing the minimization in (13) is non-trivial because the sign$(\cdot)$ function in (1) is non-continuous. We therefore introduce a smooth loss function $\mathcal{L}_V$, which mimics $\mathcal{E}_V$, but is better behaved.

$$\mathcal{L}_V(\mathbf{L}) = \frac{1}{|V|} \sum_{(\mathbf{x},y)\in V} s_a(yh(\mathbf{x})) \text{ where: } s_a(z) = \frac{1}{1 + e^{az}}. \tag{13}$$

The function $s_a(z)$ is the mirrored sigmoid function, a soft approximation of the zero-one loss. The parameter a adjusts the steepness of the curve. In the limit, as $a \gg 0$ the function $\mathcal{L}_V$ becomes identical to $\mathcal{E}_V$. Figure 1 illustrates the function $s_a(\cdot)$ for various values of a.

4.2 Gradient Computation

Our surrogate loss function $\mathcal{L}_V$ is continuous and differentiable so we can compute the derivative $\frac{\partial \mathcal{L}_V}{\partial h(\mathbf{x})}$. To obtain the derivative of $\mathcal{L}_V$ with respect to $\mathbf{L}$ we need to complete the chain-rule and also compute $\frac{\partial h(\mathbf{x})}{\partial \mathbf{L}}$. The SVM prediction function $h(\mathbf{x})$, defined in (1), depends on $\mathbf{L}$ indirectly through α_i, b and $\mathbf{K}$. In the next paragraph we follow the original approach of (Chapelle et al., 2002) for kernel parameter learning. This approach has also been used successfully for wrapper-based multiple-kernel-learning (Rakotomamonjy et al., 2008; Sonnenburg et al., 2006; Kloft et al., 2010). For ease of notation, we abbreviate $h(\mathbf{x})$ by h and use the vector notation $\alpha = [\alpha_1, \ldots, \alpha_n]^\top$. Applying the chain-rule to the derivative of h results in:

$$\frac{\partial h}{\partial \mathbf{L}} = \frac{\partial h}{\partial \alpha}\frac{\partial \alpha}{\partial \mathbf{L}} + \frac{\partial h}{\partial \mathbf{K}}\frac{\partial \mathbf{K}}{\partial \mathbf{L}} + \frac{\partial h}{\partial b}\frac{\partial b}{\partial \mathbf{L}}. \tag{14}$$

The derivatives $\frac{\partial h}{\partial \alpha}, \frac{\partial h}{\partial b}, \frac{\partial h}{\partial \mathbf{K}}, \frac{\partial \mathbf{K}}{\partial \mathbf{L}}$ are straight-forward and follow from definitions (12) and (1) (Petersen and Pedersen, 2008). In order to compute $\frac{\partial \alpha}{\partial \mathbf{L}}$ and $\frac{\partial b}{\partial \mathbf{L}}$, we express the vector (α, b) in closed-form with respect to $\mathbf{L}$. Because we absorb slack variables through our kernel modification in (3) and we use a hard-margin SVM with the modified kernel, all support vectors must lie exactly one unit from the hyperplane and satisfy

$$y_i(\sum_{j=1}^{n} \mathbf{K}_{ij}\alpha_j y_j + b) = 1. \tag{15}$$

Since the parameters α_j of non-support vectors are zero, the derivative of these α_j with respect to $\mathbf{L}$ are also all-zero and do not need to be factored into our calculation. We can

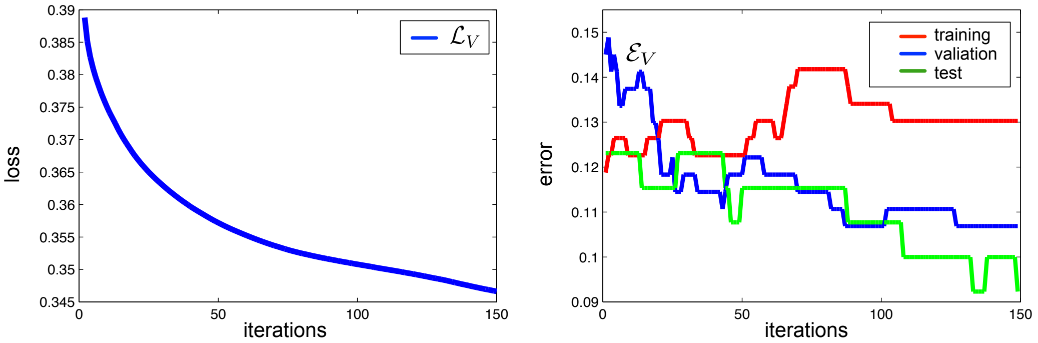

Figure 2: An example of training, validation and test error on the Credit data set. As the loss $\mathcal{L}_V$ (left) decreases, the validation error $\mathcal{E}_V$ (right) follows suit (solid blue lines). For visualization purposes, we did not use a second-order function minimizer but simple gradient descent with a small step-size.

therefore (with a slight abuse of notation) remove all rows and columns of $\mathbf{K}$ that do not correspond to support vectors and express (15) as a matrix equality

$$\underbrace{\begin{pmatrix} \bar{\mathbf{K}} & \mathbf{y} \\ \mathbf{y}^\top & 0 \end{pmatrix}}_{\mathbf{H}} \begin{pmatrix} \alpha \\ b \end{pmatrix} = \begin{pmatrix} \mathbf{1} \\ 0 \end{pmatrix}$$

where $\bar{\mathbf{K}}_{ij} = y_i y_j \mathbf{K}(x_i, x_j)$. Consequently, we can solve for α and b through left-multiplication with $\mathbf{H}^{-1}$. Further, the derivative with respect to $\mathbf{L}$ can be derived from the matrix inverse rule (Petersen and Pedersen, 2008), leading to

$$(\alpha, b)^\top = \mathbf{H}^{-1}(1 \cdots 1, 0)^\top \quad \text{and} \quad \frac{\partial(\alpha, b)}{\partial \mathbf{L}_{ij}} = -\mathbf{H}^{-1}\frac{\partial \mathbf{H}}{\partial \mathbf{L}_{ij}}(\alpha, b)^\top. \tag{16}$$

4.3 Optimization

Because the derivative $\frac{\partial \mathbf{H}}{\partial \mathbf{L}}$ follows directly from the definition of $\bar{\mathbf{K}}$ and (12), this completes the gradient $\frac{\partial \mathcal{L}_V}{\partial \mathbf{L}}$. We can now use standard gradient descent, or second order methods to minimize (13) up to a local minimum. It is important to point out that (16) requires the computation of the optimal α, b, given the current matrix $\mathbf{L}$. These can be obtained with any one of the many freely available SVM packages (Chang and Lin, 2001) by solving the SVM optimization (2) for the kernel $\mathbf{K}$ that results from $\mathbf{L}$. In addition, we also learn the regularization constant C from eq. (3) with our gradient descent optimization. For brevity we omit the exact derivation of $\frac{\partial \mathcal{L}_V}{\partial C}$ but point out that it is very similar to the gradient with respect to $\mathbf{L}$, except that it is computed only from the diagonal entries of $\mathbf{K}$.

We control the steps of gradient descent by early-stopping. We use part of the training data as a small hold-out set to monitor the algorithm's performance, and we stop the gradient descent when the validation results cease to improve.

We refer to our algorithm as *Support Vector Metric Learning* (SVML). Algorithm 1 summarizes SVML in pseudo-code. Figure 2 illustrates the value of the loss function $\mathcal{L}_V$ as well as the training, validation and test errors.

Algorithm 1 SVML in pseudo-code.

```
Initialize $\mathbf{L}$.
while Hold-out set result keeps improving do
    Compute kernel matrix $\mathbf{K}$ from $\mathbf{L}$ as in (7).
    Call SVM with $\mathbf{K}$ to obtain $\alpha$ and $b$.
    Compute gradient $\frac{\partial \mathcal{L}_V}{\partial \mathbf{L}}$ as in (16) and perform update on $\mathbf{L}$.
end while
```

4.4 Regularization and Variations

In total, we learn $d \times d$ parameters for the matrix $\mathbf{L}$ and $n+1$ parameters for α and b. To avoid overfitting, we add a regularization term to the loss function, which restricts the matrix $\mathbf{L}$ from deviating too much from its initial estimate $\mathbf{L}_0$:

$$\mathcal{L}_V(\mathbf{L}) = \frac{1}{|V|} \sum_{(\mathbf{x},y) \in V} s_a(yh(\mathbf{x})) + \lambda \|\mathbf{L} - \mathbf{L}_0\|_F^2 \tag{17}$$

Another way to avoid overfitting is to impose structural restrictions on the matrix $\mathbf{L}$. If $\mathbf{L}$ is restricted to be spherical, $\mathbf{L} = \frac{1}{\sigma}\mathbf{I}^{d \times d}$, SVML reduces to kernel width estimation. Alternatively, one can restrict $\mathbf{L}$ to be any diagonal matrix, essentially performing feature re-weighing. This can also be useful as a method for feature selection in settings with noisy features (Weston et al., 2001). We refer to these two settings as SVML-Sphere and SVML-Diag. Both of these special scenarios have been studied in previous work in the context of kernel parameter estimation (Ayat et al., 2005; Chapelle et al., 2002). See section 6 for a discussion on related work.

Another interesting structural limitation is to enforce $\mathbf{L} \in \mathcal{R}^{r \times d}$ to be rectangular, by setting $r < d$. This can be particularly useful for data visualization. For high dimensional data, the decision boundary of support machines is often hard to conceptualize. By setting $r = 2$ or $r = 3$, the data is mapped into a low dimensional space and can easily be plotted.

4.5 Implementation

The gradient, as described in this section, can be computed very efficiently. We use a simple C/Mex implementation with Matlab. As our SVM solver, we use the open-source Newton-Raphson implementation from Olivier Chapelle[3]. As function minimizer we use an open-source implementation of conjugate gradient descent[4]. Profiling of our code reveals that over 95% of the gradient computation time was spent calling the SVM solver. For a large-scale implementation, one could use special purpose SVM solvers that are optimized for speed (Bottou et al., 2007; Joachims, 1998). Also, the only computationally intensive

[3] Available at http://olivier.chapelle.cc/primal/.

[4] Courtesy of Carl Edward Rasmussen, available from `http://www.gatsby.ucl.ac.uk/~edward/code/minimize/minimize.m`

parts of the gradient outside of the SVM calls are all trivially parallelizable and could be computed on multiple cores or graphics cards. However, as it is besides the point of this paper, we do not focus on further scalability.

5. Results

To evaluate SVML, we revisit the nine data sets from Section 3.4. For convenience, Table 2 restates all relevant data statistics and also includes classification accuracies for all metric learning algorithms. SVML is naturally slower than SVM with Euclidean distance but requires no cross validation for any meta parameters. For better comparison, we also include results for 1-fold and 5-fold cross validation for all other algorithms. In both cases, the meta parameters σ^2, C were selected from five candidates each – resulting in 25 or 125 SVM executions. The kernel width σ^2 is selected from within the set $\{4d, 2d, d, \frac{d}{2}, \frac{d}{4}\}$ and the meta parameter C was chosen from within $\{0.1, 1, 10, 100\}$. As SVML is not particularly sensitive to the exact choice of λ – the regularization parameter in (17) – we set it to 100 for the smaller data sets ($n < 1000$) and to 10 for the larger ones (`Page`, `Gamma`). We terminate our algorithm based on a small hold-out set.

Statistics	**Haber**	**Credit**	**ACredit**	**Trans**	**Diabts**	**Mammo**	**CMC**	**Page**	**Gamma**
#examples	306	653	690	748	768	830	962	5743	19020
#features	3	15	14	4	8	5	9	10	11
#training exam.	245	522	552	599	614	664	770	4594	15216
#testing exam.	61	131	138	150	154	166	192	1149	3804
Metric	**Error Rates**								
Euclidean 1-fold	27.16	13.16	14.36	21.05	23.84	18.43	27.12	**2.61**	12.70
Euclidean 3-fold	27.40	13.10	14.13	20.58	23.39	18.27	26.77	**2.55**	12.68
Euclidean 5-fold	27.37	13.12	14.11	20.54	23.46	18.17	26.91	**2.56**	**12.62**
ITML + SVM 1-fold	26.57	13.78	14.15	23.01	23.19	19.14	28.65	4.82	22.63
ITML + SVM 3-fold	**26.13**	13.58	**13.88**	22.98	23.17	17.98	27.68	4.77	21.50
ITML + SVM 5-fold	26.50	13.68	14.71	22.86	23.14	18.20	27.67	4.78	21.50
NCA + SVM 1-fold	26.44	13.74	14.14	22.89	**22.84**	17.76	27.47	4.73	N/A
NCA + SVM 3-fold	26.47	13.45	14.00	22.67	**22.72**	18.12	26.60	4.73	N/A
NCA + SVM 5-fold	26.39	13.48	14.10	22.59	**22.74**	18.17	**26.53**	4.74	N/A
LMNN + SVM 1-fold	**26.38**	13.11	13.97	21.02	22.97	17.84	26.80	2.85	13.04
LMNN + SVM 3-fold	26.44	13.30	**13.93**	20.73	**22.86**	**17.57**	26.66	2.81	12.79
LMNN + SVM 5-fold	26.70	13.48	**13.89**	20.81	**22.89**	17.78	26.68	**2.66**	13.04
SVML-Sphere	27.42	13.43	**13.78**	**20.26**	23.24	17.81	28.23	3.61	12.70
SVML-Diag	28.15	13.33	15.11	20.46	24.14	**17.35**	29.51	2.92	**12.54**
SVML	**25.99**	**12.83**	**13.92**	20.89	23.25	**17.57**	**26.34**	3.41	**12.54**

Table 2: Statistics and error rates for all data sets. The data sets are sorted by smallest to largest from left to right. The table shows statistics of data sets and error rates of SVML and comparison algorithms. The best results (up to a 5% confidence interval) are highlighted in bold.

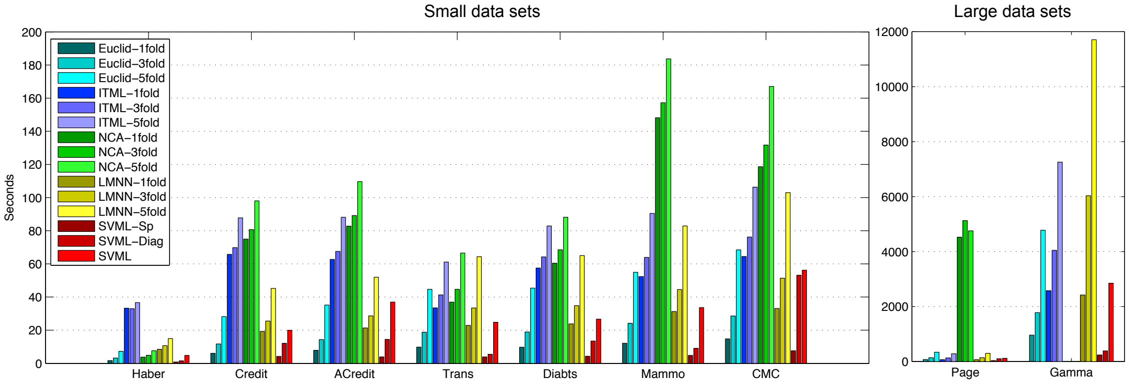

Figure 3: Timing results on all data sets. The timing includes metric learning, SVM training and cross validation. The computational resources for SVML training are roughly comparable with 3-5 fold cross validation with a Euclidean metric. (NCA did not scale to the Gamma data set.)

As in Section 2, experimental results are obtained by averaging over multiple runs on randomly generated 80/20 splits of each data set. For small data sets, we average 200 splits, 20 for medium size, and 1 for the large data set Gamma (where train/test splits are pre-defined). For the SVML training, we further apply a 50/50 split for training and validation within the training set, and another 50/50 split on the validation set for early stopping. The result from SVML appeared fairly insensitive to these splits.

As a general trend, SVML with a full matrix obtains the best results (up to significance) on 6 out of the 9 data sets. It is the only metric that consistently outperforms Euclidean distances. The diagonal version SVML-Diag and SVML-Sphere both obtain best results in 2 out of 9 and are not better than the uninformed Euclidean distance with 5-fold cross validation. None of the kNN metric learning algorithms perform comparably.

In general, we found the time required for SVML training to be roughly between 3-fold and 5-fold cross validation for Euclidean metrics, usually outperforming LMNN, ITML and NCA. Figure 3 provides running-time details on all data sets. We consider the small additional time required for SVML over Euclidean distances with cross validation as highly encouraging.

5.1 Dimensionality Reduction.

In addition to better classification results, SVML can also be used to map data into a low dimensional space while learning the SVM, allowing effective visualizations of SVM decision boundaries even for high dimensional data. To evaluate the capabilities of our algorithm for dimensionality reduction and visualization, we restrict $\mathbf{L}$ to be rectangular. Specifically, a mapping into a $r = 2$ or $r = 3$ dimensional space. As comparison, we use PCA to reduce the dimensionality before the SVM training without SVML (all meta parameters were set by cross-validation). Figure 4 shows the visualization of the support vectors of the Credit data set after a mapping into a two dimensional space with SVML

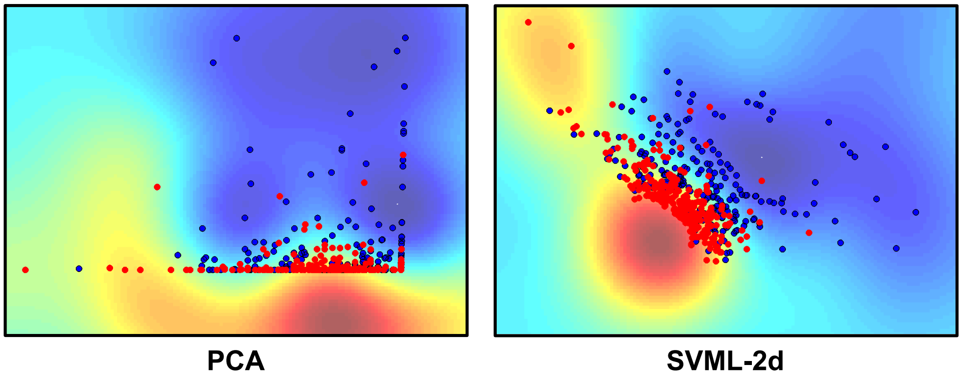

Figure 4: $2D$ visualization of the Credit data set. The figure shows the decision surface and support vectors generated by SVML ($\mathbf{L} \in \mathcal{R}^{2\times d}$) and standard SVM after projection onto the two leading principal components.

and PCA. The background is colored by the prediction function $h(\cdot)$. The $2D$ visualization shows a much more interpretable decision boundary. (Visualizations of the LMNN and NCA mappings were very similar to those of PCA.) Visualizing the support vectors and the decision boundaries of kernelized SVMs can help demystify hyperplanes in reproducing kernel Hilbert spaces and might help with data analysis.

6. Related Work

Multiple publications introduce methods to learn Mahalanobis metrics. Previous work has focussed primarily on Mahalanobis metrics for k-nearest neighbor classifiers (Davis et al., 2007; Globerson and Roweis, 2005; Goldberger et al., 2005; Shental et al., 2002; Shalev-Shwartz et al., 2004; Weinberger et al., 2006) and clustering (Davis et al., 2007; Shalev-Shwartz et al., 2004; Shental et al., 2002; Xing et al., 2002). None of these algorithms is specifically geared towards SVM classification. A detailed discussion of NCA, ITML and LMNN is provided in Section 3.

Another related line of work focusses on learning of the kernel matrix. The most common approach is to find convex combinations of already existing kernel matrices (Bach et al., 2004; Lanckriet et al., 2004) or kernel learning through semi-definite programming (Graepel, 2002; Ong et al., 2005). The most similar area of related work is the field of kernel parameter estimation (Ayat et al., 2005; Chapelle et al., 2002; Cherkassky and Ma, 2004; Friedrichs and Igel, 2005). In particular, (Friedrichs and Igel, 2005) can be viewed as learning a Mahalanobis metric for the Gaussian kernel – however, instead of minimizing a soft surrogate of the validation error with gradient descent, the authors use genetic programming to maximize the “fittness” of the kernel parameters. The method of (Chapelle et al., 2002) uses gradient descent to learn the σ parameter of the RBF kernel matrix. SVML was highly inspired by this work. The main difference between our work and (Chapelle et al., 2002) is that SVML learns the full matrix $\mathbf{L}$, and therefore a Mahalanobis metric, whereas Chapelle et al. only learn the parameter σ or individual weights for blocks of features. Spherical and diagonal SVML can be viewed as a version of (Chapelle et al., 2002). Similarly, (Ayat et al., 2005; Schittkowski, 2005) also explore feature re-weighting for support vector machines with alternative loss functions.

7. Conclusion

In this paper we investigate metric learning for SVMs. An empirical study of three of the most widely used out-of-the-box metric learning algorithms for kNN classification shows that these are not particularly well suited for SVMs. As an alternative, we derive SVML, an algorithm that seamlessly combines support vector classification with distance metric learning. SVML learns a metric that attempts to minimize the validation error of the SVM prediction at the same time as it trains the SVM classifier. On several standard benchmark datasets we demonstrate that our algorithm achieves state-of-the-art results with very high reliability. An important feature of SVML is that it is very insensitive to its few parameters (which we all set to default values) and does not require any model selection by cross validation. In fact, we demonstrate that SVML outperforms traditional SVM-RBF with the Euclidean distance (where parameters are set through cross validation) consistently in accuracy while requiring a comparable amount of computation time. These aspects make SVML a very promising general-purpose metric learning algorithm for SVMs with RBF kernels, which also incorporates automatic model selection. We are currently implementing an open-source plug-in for the popular LIBSVM library (Chang and Lin, 2001) and extending it to multi-class settings.

7.1 Acknowledgements

We thank Marius Kloft, Ulrich Rueckert, Cheng Soon Ong, Alain Rakotomamonjy, Soeren Sonnenburg and Francis Bach for motivating this work. We also thank the National Institute of Health for supporting Zhixiang Xu on grant NIH 1-U01-NS073457-01. Further, we would like to thank Yahoo Research for their generous support through the Yahoo Research Faculty Engagement Program.

References

N. E. Ayat, M. Cheriet, and C. Y. Suen. Automatic model selection for the optimization of SVM kernels. *Pattern Recognition*, 38(10):1733–1745, 2005.

F.R. Bach, G.R.G. Lanckriet, and M.I. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the twenty-first international conference on Machine learning*, page 6. ACM, 2004.

L. Bottou, O. Chapelle, and D. DeCoste. *Large-scale kernel machines*. MIT Press, 2007.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge, England, 2004.

C. Chang and C. Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46(1):131–159, 2002.

V. Cherkassky and Y. Ma. Practical selection of SVM parameters and noise estimation for SVM regression. *Neural Networks*, 17(1):113–126, 2004.

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

T. Cover and P. Hart. Nearest neighbor pattern classification. In *IEEE Transactions in Information Theory, IT-13*, pages 21–27, 1967.

J.V. Davis, B. Kulis, P. Jain, S. Sra, and I.S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. URL `http://archive.ics.uci.edu/ml`.

F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64: 107–117, 2005.

A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems 18*, 2005.

J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 513–520, Cambridge, MA, 2005. MIT Press.

T. Graepel. Kernel matrix completion by semidefinite programming. *Artificial Neural Networks ICANN 2002*, pages 141–142, 2002.

T. Joachims. Making large-scale svm learning practical. LS8-Report 24, Universität Dortmund, LS VIII-Report, 1998.

M. Kloft, U. Brefeld, S. Sonnenburg, and A. Zien. Non-sparse regularization and efficient training with multiple kernels. *Arxiv preprint arXiv:1003.0079*, 2010.

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

S.P. Lloid. Least squares quantization in PCM. *Special issue on quantization of the IEEE trans. on information theory*, 1982.

P.C. Mahalanobis. On the generalized distance in statistics. In *Proceedings of the National Institute of Science, Calcutta*, volume 12, page 49, 1936.

C.S. Ong, A.J. Smola, and R.C. Williamson. Learning the kernel with hyperkernels. *Journal of Machine Learning Research*, 6(07), 2005.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, oct 2008. URL `http://www2.imm.dtu.dk/pubdb/p.php?3274`. Version 20081110.

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

K. Schittkowski. Optimal parameter selection in support vector machines. *Journal of Industrial and Management Optimization*, 1(4):465, 2005.

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

S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the 21st International Conference on Machine Learning*, Banff, Canada, 2004.

N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment learning and relevant component analysis. In *Proceedings of the Seventh European Conference on Computer Vision (ECCV-02)*, volume 4, pages 776–792, London, UK, 2002. Springer-Verlag. ISBN 3-540-43748-7.

S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *The Journal of Machine Learning Research*, 7:1531–1565, 2006. ISSN 1532-4435.

I. Steinwart. On the influence of the kernel on the consistency of support vector machines. *Journal of Machine Learning Research*, 2:67–93, 2002. ISSN 1532-4435.

V. Vapnik. *Statistical Learning Theory.* Wiley, N.Y., 1998.

K. Q. Weinberger, J. C. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. MIT Press, 2006.

J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for SVMs. *Advances in neural information processing systems*, pages 668–674, 2001.

E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.