Fast Solvers and Efficient Implementations for Distance Metric Learning

Kilian Q. Weinberger KILIAN@YAHOO-INC.COM
Yahoo! Research, 2821 Mission College Blvd, Santa Clara, CA 9505

Lawrence K. Saul SAUL@CS.UCSD.EDU
CSE Department, University of California, San Diego 9500 Gilman Drive, La Jolla, CA 92093-0404

Abstract

In this paper we study how to improve nearest neighbor classification by learning a Mahalanobis distance metric. We build on a recently proposed framework for distance metric learning known as large margin nearest neighbor (LMNN) classification. Our paper makes three contributions. First, we describe a highly efficient solver for the particular instance of semidefinite programming that arises in LMNN classification; our solver can handle problems with billions of large margin constraints in a few hours. Second, we show how to reduce both training and testing times using metric ball trees; the speedups from ball trees are further magnified by learning low dimensional representations of the input space. Third, we show how to learn different Mahalanobis distance metrics in different parts of the input space. For large data sets, the use of locally adaptive distance metrics leads to even lower error rates.

1. Introduction

Many algorithms for pattern classification and machine learning depend on computing distances in a multidimensional input space. Often, these distances are computed using a Euclidean distance metric—a choice which has both the advantages of simplicity and generality. Notwithstanding these advantages, though, the Euclidean distance metric is not very well adapted to most problems in pattern classification.

Viewing the Euclidean distance metric as overly simplistic, many researchers have begun to ask how to learn or adapt the distance metric itself in order to achieve better results (Xing et al., 2002; Chopra et al., 2005; Frome et al., 2007). Distance metric learning is an emerging area of statistical learning in which the goal is to induce a more powerful distance metric from labeled examples. The simplest instance of this problem arises in the context of k-nearest neighbor (kNN) classification using Mahalanobis distances. Mahalanobis distances are computed by linearly transforming the input space, then computing Euclidean distances in the transformed space. A well-chosen linear transformation can improve kNN classification by decorrelating and reweighting elements of the feature vector. In fact, significant improvements have been observed within several different frameworks for this problem, including neighborhood components analysis (Goldberger et al., 2005), large margin kNN classification (Weinberger et al., 2006), and information-theoretic metric learning (Davis et al., 2007).

These studies have established the general utility of distance metric learning for kNN classification. However, further work is required to explore its promise in more difficult regimes. In particular, larger data sets raise new and important challenges in scalability. They also present the opportunity to learn more adaptive and sophisticated distance metrics.

In this paper, we study these issues as they arise in the recently proposed framework of large margin nearest neighbor (LMNN) classification (Weinberger et al., 2006). In this framework, a Mahalanobis distance metric is trained with the goal that the k-nearest neighbors of each example belong to the same class while examples from different classes are separated by a large margin. Simple in concept, useful in practice, the ideas behind LMNN classification have also inspired other related work in machine learning and computer vision (Torresani & Lee, 2007; Frome et al., 2007).

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008.

The role of the margin in LMNN classification is inspired by its role in support vector machines (SVMs). Not surprisingly, given these roots, LMNN classification also inherits various strengths and weaknesses of SVMs (Schölkopf & Smola, 2002). For example, as in SVMs, the training procedure in LMNN classification reduces to a convex optimization based on the hinge loss. However, as described in section 2, naïve implementations of this optimization do not scale well to larger data sets.

Addressing the challenges and opportunities raised by larger data sets, this paper makes three contributions. First, we describe how to optimize the training procedure for LMNN classification so that it can readily handle data sets with tens of thousands of training examples. In order to scale to this regime, we have implemented a special-purpose solver for the particular instance of semidefinite programming that arises in LMNN classification. In section 3, we describe the details of this solver, which we have used to tackle problems involving billions of large margin constraints. To our knowledge, problems of this size have yet to be tackled by other recently proposed methods (Goldberger et al., 2005; Davis et al., 2007) for learning Mahalanobis distance metrics.

As the second contribution of this paper, we explore the use of metric ball trees (Liu et al., 2005) for LMNN classification. These data structures have been widely used to accelerate nearest neighbor search. In section 4, we show how similar data structures can be used for faster training and testing in LMNN classification. Ball trees are known to work best in input spaces of low to moderate dimensionality. Mindful of this regime, we also show how to modify the optimization in LMNN so that it learns a low-rank Mahalanobis distance metric. With this modification, the metric can be viewed as projecting the original inputs into a lower dimensional space, yielding further speedups.

As the third contribution of this paper, we describe an important extension to the original framework for LMNN classification. Specifically, in section 5, we show how to learn different Mahalanobis distance metrics for different parts of the input space. The novelty of our approach lies in learning a collection of different local metrics to maximize the margin of correct kNN classification. The promise of this approach is suggested by recent, related work in computer vision that has achieved state-of-the-art results on image classification (Frome et al., 2007). Our particular approach begins by partitioning the training data into disjoint clusters using class labels or unsupervised methods. We then learn a Mahalanobis distance metric for each cluster. While the training procedure couples the distance metrics in different clusters, the optimization remains a convex problem in semidefinite programming. The globally coupled training of these metrics also distinguishes our approach from earlier work in adaptive distance metrics for kNN classification (Hastie & Tibshirani, 1996). To our knowledge, our approach yields the best kNN test error rate on the extensively benchmarked MNIST data set of handwritten digits that does not incorporate domain-specific prior knowledge (LeCun et al., 1998; Simard et al., 1993). Thus, our results show that we can exploit large data sets to learn more powerful and adaptive distance metrics for kNN classification.

2. Background

Of the many settings for distance metric learning, the simplest instance of the problem arises in the context of kNN classification using Mahalanobis distances. A Mahalanobis distance metric computes the squared distances between two points $\vec{x}_i$ and $\vec{x}_j$ as:

$$d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j), \qquad (1)$$

where $\mathbf{M} \succeq 0$ is a positive semidefinite matrix. When $\mathbf{M}$ is equal to the identity matrix, eq. (1) reduces to the Euclidean distance metric. In distance metric learning, the goal is to discover a matrix $\mathbf{M}$ that leads to lower kNN error rates than the Euclidean distance metric.

Here we briefly review how Mahalanobis distance metrics are learned for LMNN classification (Weinberger et al., 2006). Let the training data consist of n labeled examples $\{(\vec{x}_i, y_i)\}_{i=1}^n$ where $\vec{x}_i \in \mathcal{R}^d$ and $y_i \in \{1, \ldots, c\}$, where c is the number of classes. For LMNN classification, the training procedure has two steps. The first step identifies a set of k similarly labeled *target neighbors* for each input $\vec{x}_i$. Target neighbors are selected by using prior knowledge (if available) or by simply computing the k nearest (similarly labeled) neighbors using Euclidean distance. We use the notation $j \rightsquigarrow i$ to indicate that $\vec{x}_j$ is a target neighbor of $\vec{x}_i$. The second step adapts the Mahalanobis distance metric so that these target neighbors are closer to $\vec{x}_i$ than all other differently labeled inputs. The Mahalanobis distance metric is estimated by solving a problem in semidefinite programming. Distance metrics obtained in this way were observed to yield consistent and significant improvements in kNN error rates.

The semidefinite program in LMNN classification arises from an objective function which balances two terms. The first term penalizes large distances between inputs and their target neighbors. The second term penalizes small distances between differently la-

beled inputs; specifically, a penalty is incurred if these distances do not exceed (by a finite margin) the distances to the target neighbors of these inputs. The terms in the objective function can be made precise with further notation. Let $y_{ij} \in \{0,1\}$ indicate whether the inputs $\vec{x}_i$ and $\vec{x}_j$ have the same class label. Also, let $\xi_{ijl} \geq 0$ denote the amount by which a differently labeled input $\vec{x}_l$ invades the "perimeter" around input $\vec{x}_i$ defined by its target neighbor $\vec{x}_j$. The Mahalanobis distance metric $\mathbf{M}$ is obtained by solving the following semidefinite program:

$$\begin{array}{ll} \textbf{Minimize} & \sum_{j \rightsquigarrow i} \left[d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) + \mu \sum_l (1 - y_{il}) \xi_{ijl} \right] \\ \textbf{subject to:} & \\ \textbf{(a)} & d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_l) - d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) \;\geq\; 1 - \xi_{ijl} \\ \textbf{(b)} & \xi_{ijl} \geq 0 \\ \textbf{(c)} & \mathbf{M} \succeq 0. \end{array}$$

The constant μ defines the trade-off between the two terms in the objective function; in our experiments, we set $\mu = 1$. The constraints of type **(a)** encourage inputs ($\vec{x}_i$) to be at least one unit closer to their k target neighbors ($\vec{x}_j$) than to any other differently labeled input ($\vec{x}_l$). When differently labeled inputs $\vec{x}_l$ invade the local neighborhood of $\vec{x}_i$, we refer to them as *impostors*. Impostors generate positive slack variables ξ_{ijl} which are penalized in the second term of the objective function. The constraints of type **(b)** enforce nonnegativity of the slack variables, and the constraint **(c)** enforces that $\mathbf{M}$ is positive semidefinite, thus defining a valid (pseudo)metric. Noting that the squared Mahalanobis distances $d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j)$ are linear in the matrix $\mathbf{M}$, the above optimization is easily recognized as an semidefinite program.

3. Solver

The semidefinite program in the previous section grows in complexity with the number of training examples (n), the number of target neighbors (k), and the dimensionality of the input space (d). In particular, the objective function is optimized with respect to $O(kn^2)$ large margin constraints of type **(a)** and **(b)**, while the Mahalanobis distance metric $\mathbf{M}$ itself is a $d \times d$ matrix. Thus, for even moderately large and/or high dimensional data sets, the required optimization (though convex) cannot be solved by standard off-the-shelf packages (Borchers, 1999).

In order to tackle larger problems in LMNN classification, we implemented our own special-purpose solver. Our solver was designed to exploit the particular structure of the semidefinite program in the previous section. The solver iteratively re-estimates the Mahalanobis distance metric $\mathbf{M}$ to minimize the objective function for LMNN classification. The amount of computation is minimized by careful book-keeping from one iteration to the next. The speed-ups from these optimizations enabled us to work comfortably on data sets with up to $n = 60,000$ training examples.

Our solver works by eliminating the slack variables ξ_{ijl} from the semidefinite program for LMNN classification, then minimizing the resulting objective function by sub-gradient methods. The slack variables are eliminated by folding the constraints **(a)** and **(b)** into the objective function as a sum of "hinge" losses. The hinge function is defined as $[z]_+ = z$ if $z > 0$ and $[z]_+ = 0$ if $z < 0$. In terms of this hinge function, we can express ξ_{ijl} as a function of the matrix $\mathbf{M}$:

$$\xi_{ijl}(\mathbf{M}) \;=\; \left[1 + d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_l)\right]_+ \qquad (2)$$

Finally, writing the objective function only in terms of the matrix $\mathbf{M}$, we obtain:

$$\varepsilon(\mathbf{M}) = \sum_{j \rightsquigarrow i} \left[d^2_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) + \mu \sum_l (1 - y_{il}) \xi_{ijl}(\mathbf{M}) \right]. \qquad (3)$$

This objective function is not differentiable due to the hinge losses that appear in eq. (2). Nevertheless, because it is convex, we can compute its sub-gradient and use standard descent algorithms to find its minimum. At each iteration of our solver, the optimization takes a step along the sub-gradient to reduce the objective function, then projects the matrix $\mathbf{M}$ back onto the cone of positive semidefinite matrices. Iterative methods of this form are known to converge to the correct solution, provided that the gradient step-size is sufficiently small (Boyd & Vandenberghe, 2004).

The gradient computation can be done most efficiently by careful book-keeping from one iteration to the next. As simplifying notation, let $\mathbf{C}_{ij} = (\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_j)^\top$. In terms of this notation, we can express the squared Mahalanobis distances in eq. (8) as:

$$d_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = \mathrm{tr}(\mathbf{C}_{ij}\mathbf{M}). \qquad (4)$$

To evaluate the gradient, we denote the matrix $\mathbf{M}$ at the t^{th} iteration as $\mathbf{M}^t$. At each iteration, we also define a set $\mathcal{N}^t$ of triplet indices such that $(i,j,l) \in \mathcal{N}^t$ if and only if the triplet's corresponding slack variable exceeds zero: $\xi_{ijl}(\mathbf{M}^t) > 0$. With this notation, we can write the gradient $\mathbf{G}^t = \frac{\partial \varepsilon}{\partial \mathbf{M}}\big|_{\mathbf{M}^t}$ at the t^{th} iteration as:

$$\mathbf{G}^t \;=\; \sum_{j \rightsquigarrow i} \mathbf{C}_{ij} + \mu \sum_{(i,j,l) \in \mathcal{N}^t} (\mathbf{C}_{ij} - \mathbf{C}_{il}). \qquad (5)$$

Computing the gradient requires computing the outer products in $\mathbf{C}_{ij}$; it thus scales quadratically in the

input dimensionality. As the set $\mathcal{N}^t$ is potentially large, a naïve computation of the gradient would be extremely expensive. However, we can exploit the fact that the gradient contribution from each active triplet (i, j, l) does not depend on the degree of its margin violation. Thus, the changes in the gradient from one iteration to the next are determined entirely by the differences between the sets $\mathcal{N}^t$ and $\mathcal{N}^{t+1}$. Using this fact, we can derive an extremely efficient update that relates the gradient at one iteration to the gradient at the previous one. The update subtracts the contributions from triples that are no longer active and adds the contributions from those that just became active:

$$\mathbf{G}^{t+1} = \mathbf{G}^t - \mu \sum_{(i,j,l)\in\mathcal{N}^t-\mathcal{N}^{t+1}} (\mathbf{C}_{ij} - \mathbf{C}_{il}) + \mu \sum_{(i,j,l)\in\mathcal{N}^{t+1}-\mathcal{N}^t} (\mathbf{C}_{ij} - \mathbf{C}_{il}) \tag{6}$$

For small gradient step sizes, the set $\mathcal{N}^t$ changes very little from one iteration to the next. In this case, the right hand side of eq. (6) can be computed very fast.

To further accelerate the solver, we adopt an active set method. This method is used to monitor the large margin constraints that are actually violated. Note that computing the set $\mathcal{N}^t$ at each iteration requires checking every triplet (i, j, l) with $j \rightsquigarrow i$ for a potential margin violation. This computation scales as $O(nd^2 + kn^2d)$, making it impractical for large data sets. To avoid this computational burden, we observe that the great majority of triples do not incur margin violations: in particular, for each training example, only a very small fraction of differently labeled examples typically lie nearby in the input space. Consequently, a useful approximation is to check only a subset of likely triples for margin violations per gradient computation and only occasionally perform the full computation. We set this active subset to the list of all triples that have ever violated the margin, ie $\bigcup_{i=1}^{t-1} \mathcal{N}^i$. When the optimization converges, we verify that the working set $\mathcal{N}^t$ does contain all active triples that incur margin violations. This final check is needed to ensure convergence to the correct minimum. If the check is not satisfied, the optimization continues with the newly expanded active set.

Table 1 shows how quickly the solver works on problems of different sizes. The results in this table were generated by learning a Mahalanobis distance metric on the MNIST data set of 28×28 grayscale handwritten digits (LeCun et al., 1998). The digits were preprocessed by principal component analysis (PCA) to reduce their dimensionality from $d = 784$ to $d = 169$. We experimented by learning a distance metric from different subsets of the training examples. The experiments were performed on a standard desktop machine with a 2.0 GHz dual core 2 processor. For each experiment, the table shows the number of training examples, the CPU time to converge, the number of active constraints, the total number of constraints, and the kNN test error (with $k = 3$). Note that for the full MNIST training set, the semidefinite program has over three billion large margin constraints. Nevertheless, the active set method converges in less than four hours—from a Euclidean distance metric with 2.33% test error to a Mahalanobis distance metric with 1.72% test error.

N	time	|active set|	|total set|	train error	test error
60	**9s**	**844**	**3.2K**	**0%**	**29.37%**
600	**37s**	**6169**	**323K**	**0%**	**10.79%**
6000	**4m**	**50345**	**32M**	**0.48%**	**3.13%**
60000	**3h25m**	**540037**	**3.2B**	**0%**	**1.72%**

Table 1. Statistics of the solver on subsets of the data set of MNIST handwritten digits. See text for details.

4. Tree-Based Search

Nearest neighbor search can be accelerated by storing training examples in hierarchical data structures (Liu et al., 2005). These data structures can also be used to reduce the training and test times for LMNN classification. In this section, we describe how these speedups are obtained using metric ball trees.

4.1. Ball trees

We begin by reviewing the use of ball trees (Liu et al., 2005) for fast kNN search. Ball trees recursively partition the training inputs by projecting them onto directions of large variance, then splitting the data on the mean or median of these projected values. Each subset of data obtained in this way defines a hypersphere (or "ball") in the multidimensional input space that encloses its training inputs. The distance to such a hypersphere can be easily computed for any test input; moreover, this distance provides a lower bound on the test input's distance to any of the enclosed training inputs. This bound is illustrated in Fig. 1. Let S be the set of training inputs inside a specific ball with radius r. The distance from a test input $\vec{x}_t$ to any training input $\vec{x}_i \in S$ is bounded from below by:

$$\forall \vec{x}_i \in S \quad \|\vec{x}_t - \vec{x}_i\| \geq \max(\|\vec{x}_t - \vec{c}\|_2 - r, 0). \tag{7}$$

These bounds can be exploited in a tree-based search for nearest neighbors. In particular, if the distance to the currently k^{th} closest input $\vec{x}_j$ is smaller than the bound from eq. (7), then all inputs inside the ball S can be pruned away. This pruning of unexplored subtrees can significantly accelerate kNN search. The same basic strategy can also be applied to kNN search

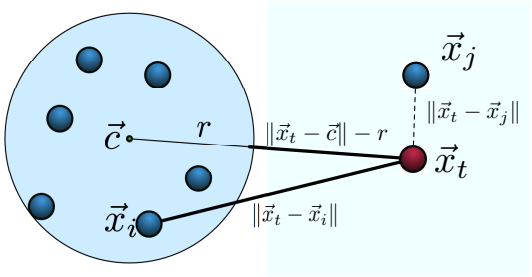

Figure 1. How ball trees work: for any input $\vec{x}_i \in S$, the distance $\|\vec{x}_t - \vec{x}_i\|$ is bounded from below by eq. (7) . If a training example $\vec{x}_j$ is known to be closer to $\vec{x}_t$, then the inputs inside the ball can be ruled out as nearest neighbors.

under a Mahalanobis distance metric.

4.2. Speedups

We first experimented with ball trees to reduce the *test* times for LMNN classification. In our experiments, we observed a factor of $3x$ speed-up for 40-dimensional data and a factor of $15x$ speedup for 15-dimensional data. Note that these speedups were measured relative to a highly optimized baseline implementation of kNN search. In particular, our baseline implementation rotated the input space to align its coordinate axes with the principal components of the data; the coordinate axes were also sorted in decreasing order of variance. In this rotated space, distance computations were terminated as soon as any partially accumulated results (from leading principal components) exceeded the currently smallest k distances from the kNN search in progress.

We also experimented with ball trees to reduce the *training* times for LMNN classification. To reduce training times, we integrated ball trees into our special-purpose solver. Specifically, ball trees were used to accelerate the search for so-called "impostors". Recall that for each training example $\vec{x}_i$ and for each of its similarly labeled target neighbors $\vec{x}_j$, the impostors consist of all differently labeled examples $\vec{x}_l$ with $d_M(\vec{x}_i, \vec{x}_l)^2 \leq d_M(\vec{x}_i, \vec{x}_j)^2 + 1$. The search for impostors dominates the computation time in the training procedure for LMNN classification. To reduce the amount of computation, the solver described in section 3 maintains an active list of previous margin violations. Nevertheless, the overall computation still scales as $O(n^2 d)$, which can be quite expensive. Note that we only need to search for impostors among training examples with different class labels. To speed up training, we built one ball tree for the training examples in each class and used them to search for impostors (as the ball-tree creation time is negligible in comparison with the impostor search, we re-built it in every iteration). We observed the ball trees to yield speedups ranging from a factor of $1.9x$ with 10-dimensional data to a factor of $1.2x$ with 100 dimensional data.

4.3. Dimensionality reduction

Across all our experiments, we observed that the gains from ball trees diminished rapidly with the dimensionality of the input space. This observation is consistent with previous studies of ball trees and NN search. When the data is high dimensional, NN search is plagued by the so-called "curse of dimensionality". In particular, distances in high dimensions tend to be more uniform, thereby reducing the opportunities for pruning large subtrees.

The curse of dimensionality is often addressed in ball trees by projecting the stored training inputs into a lower dimensional space. The most commonly used methods for dimensionality reduction are random projections and PCA. Despite their widespread use, however, neither of these methods is especially geared to preserve (or improve) the accuracy of kNN classification.

We experimented with two methods for dimensionality reduction in the particular context of LMNN classification. Both methods were based on learning a low-rank Mahalanobis distance metric. Such a metric can be viewed as projecting the original inputs into a lower dimensional space. In our first approach, we performed a singular value decomposition (SVD) on the full rank solution to the semidefinite program in section 2. The full rank solution for the distance metric was then replaced by a low rank approximation based on its leading eigenvectors. We call this approach LMNN-SVD. In our second approach, we followed a suggestion from previous work on LMNN classification (Torresani & Lee, 2007). In this approach, we explicitly parameterized the Mahalanobis distance metric as a low-rank matrix, writing $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, where $\mathbf{L}$ is a rectangular matrix. To obtain the distance metric, we optimized the same objective function as before, but now in terms of the explicitly low-rank linear transformation $\mathbf{L}$. The optimization over $\mathbf{L}$ is not convex unlike the original optimization over $\mathbf{M}$, but a (possibly local) minimum can be computed by standard gradient-based methods. We call this approach LMNN-RECT.

Fig. 2 shows the results of kNN classification from both these methods on the MNIST data set of handwritten digits. For these experiments, the raw MNIST images (of size 28× 28) were first projected onto their 350 leading principal components. The training procedure for LMNN-SVD optimized a full-rank distance metric in this 350 dimensional space, then extracted a low-rank distance metric from its leading eigenvectors.

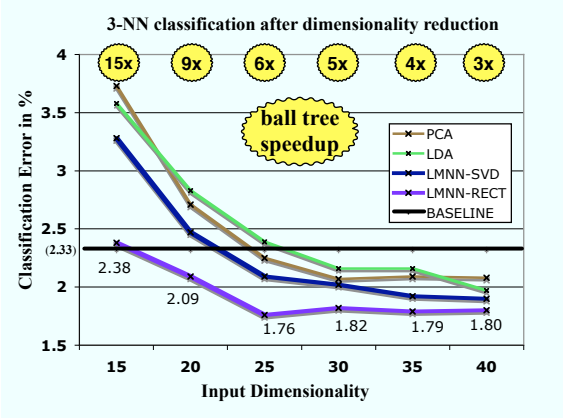

Figure 2. Graph of kNN error rate (with $k = 3$) on different low dimensional representations of the MNIST data set.

The training procedures for LMNN-RECT optimized a low-rank rectangular matrix of size $r \times 350$, where r varied from 15 to 40. Also shown in the figure are the results from further dimensionality reduction using PCA, as well as the baseline kNN error rate in the original (high dimensional) space of raw images. The speedup from ball trees is shown at the top of the graph. The amount of speedup depends significantly on the amount of dimensionality reduction, but very little on the particular method of dimensionality reduction. Of the three methods compared in the figure, LMNN-RECT is the most effective, improving significantly over baseline kNN classification while operating in a much lower dimensional space. Overall, these results show that aggressive dimensionality reduction can be combined with distance metric learning.

5. Multiple Metrics

The originally proposed framework for LMNN classification has one clear limitation: the same Mahalanobis distance metric is used to compute distances everywhere in the input space. Writing the metric as $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$, we see that Mahalanobis distances are equivalent to Euclidean distances after a global linear transformation $\vec{x} \rightarrow \mathbf{L}\vec{x}$ of the input space. Such a transformation cannot adapt to nonlinear variabilities in the training data.

In this section, we describe how to learn different Mahalanobis distance metrics in different parts of the input space. We begin by simply describing how such a collection of local distance metrics is used at test time. Assume that the data set is divided into p disjoint partitions $\{P_\alpha\}_{\alpha=1}^p$, such that $P_\alpha \cap P_\beta = \{\}$ for any $\alpha \neq \beta$ and $\bigcup_\alpha P_\alpha = \{\vec{x}_i\}_{i=1}^n$. Also assume that each partition P_α has its own Mahalanobis distance metric $\mathbf{M}_\alpha$ for use in kNN classification. Given a test vector $\vec{x}_t$, we compute its squared distance to a training input $\vec{x}_i$ in partition α_i as:

$$d^2_{\mathbf{M}_{\alpha_i}}(\vec{x}_t, \vec{x}_i) = (\vec{x}_t - \vec{x}_i)^\top \mathbf{M}_{\alpha_i}(\vec{x}_t - \vec{x}_i). \qquad (8)$$

These distances are then sorted as usual to determine nearest neighbors and label the test input. Note, however, how different distance metrics are used for training inputs in different partitions.

We can also use these metrics to compute distances between training inputs, with one important caveat. Note that for inputs belonging to different partitions, the distance between them will depend on the particular metric used to compute it. This asymmetry does not present any inherent difficulty since, in fact, the dissimilarity measure in kNN classification is not required to be symmetric. Thus, even on the training set, we can use multiple metrics to measure distances and compute meaningful leave-one-out kNN error rates.

5.1. Learning algorithm

In this section we describe how to learn multiple Mahalanobis distance metrics for LMNN classification. Each of these metrics is associated with a particular cluster of training examples. To derive these clusters, we experimented with both unsupervised methods, such as the k-means algorithm, and fully supervised methods, in which each cluster contains the training examples belonging to a particular class.

Before providing details of the learning algorithm, we make the following important observation. Multiple Mahalanobis distance metrics for LMNN classification cannot be learned in a decoupled fashion—that is, by solving a collection of simpler, independent problems of the type already considered (e.g., one within each partition of training examples). Rather, the metrics must be learned in a coordinated fashion so that the distances from different metrics can be meaningfully compared for kNN classification. In our framework, such comparisons arise whenever an unlabeled test example has potential nearest neighbors in two or more different clusters of training examples.

Our learning algorithm for multiple local distance metrics $\{\mathbf{M}_\alpha\}_{\alpha=1}^p$ generalizes the semidefinite program for ordinary LMNN classification in section 2. First, we modify the objective function so that the distances to target neighbors $\vec{x}_j$ are measured under the metric $\mathbf{M}_{\alpha_j}$. Next, we modify the large margin constraints in **(a)** so that the distances to potential impostors $\vec{x}_l$ are measured under the metric $\mathbf{M}_{\alpha_l}$. Finally, we replace the single positive semidefinite constraint in **(c)**

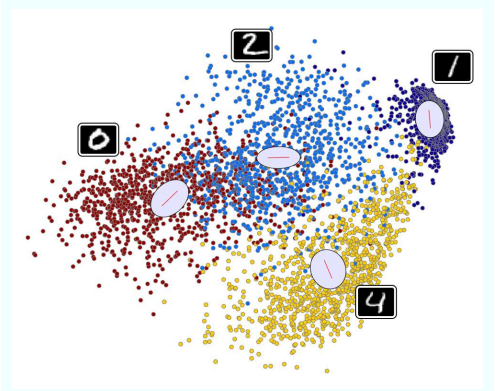

Figure 3. Visualization of multiple local distance metrics for MNIST handwritten digits. See text for details.

by multiple such constraints, one for each local metric $\mathbf{M}_\alpha$. Taken together, these steps lead to the new semidefinite program:

Minimize $\sum_{j \rightsquigarrow i} \left[d^2_{\mathbf{M}_{\alpha_j}}(\vec{x}_i, \vec{x}_j) + \mu \sum_l (1 - y_{il}) \xi_{ijl} \right]$
subject to:
(a) $d^2_{\mathbf{M}_{\alpha_l}}(\vec{x}_i, \vec{x}_l) - d^2_{\mathbf{M}_{\alpha_j}}(\vec{x}_i, \vec{x}_j) \;\geq\; 1 - \xi_{ijl}$
(b) $\xi_{ijl} \geq 0$
(c) $\mathbf{M}_\alpha \succeq 0$.

Note how the new constraints in **(a)** couple the different Mahalanobis distance metrics. By jointly optimizing these metrics to minimize a single objective function, we ensure that the distances they compute can be meaningfully compared for kNN classification.

5.2. Results

We evaluated the performance of this approach on five publicly available data sets: the MNIST data set[1] of handwritten digits ($n = 60000$, $c = 10$), the 20-Newsgroups data set[2] of text messages ($n = 18827$, $c = 20$), the Letters data set[3] of distorted computer fonts ($n = 14000$, $c = 26$), the Isolet data set[4] of spoken letters ($n = 6238$, $c = 26$), and the YaleFaces[5] data set of face images ($n = 1690$, $c = 38$). The data sets were preprocessed by PCA to reduce their dimensionality. The amount of dimensionality reduction varied with each experiment, as discussed below.

To start, we sought to visualize the multiple metrics learned in a simple experiment on MNIST handwritten digits of zeros, ones, twos, and fours. For ease of visualization, we worked with only the leading two principal components of the MNIST data set. Fig. 3 shows these two dimensional inputs, color-coded by class label. With these easily visualized inputs, we minimized the objective function in section 5.1 to learn a specialized distance metric for each type of handwritten digit. The ellipsoids in the plot reveal the directions amplified by the local distance metric of each digit class. Notably, each distance metric learns to amplify the direction perpendicular to the decision boundary for the nearest, competing class of digits.

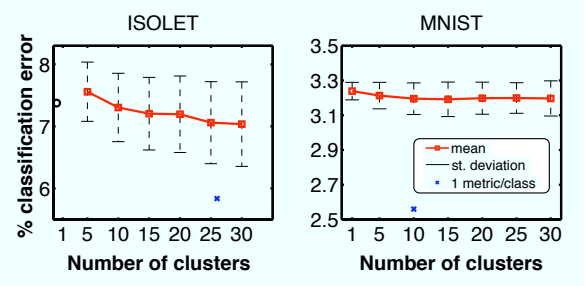

Figure 4. Test kNN error rates on the Isolet and MNIST data sets as a function of the number of distance metrics.

Our next experiments examined the performance of LMNN classification as a function of the number of distance metrics. In these experiments, we used PCA to reduce the input dimensionality to $d = 50$; we also only worked with a subset of $n = 10000$ training examples of MNIST handwritten digits. To avoid overfitting, we used an "early stopping" approach while monitoring the kNN error rates on a held-out validation set consisting of 30% of the training data.

Fig. 4 shows the test kNN error rates on the Isolet and MNIST data sets as a function of the number of distance metrics. In these experiments, we explored both unsupervised and supervised methods for partitioning the training inputs as a precursor to learning local distance metrics. In the unsupervised setting, the training examples were partitioned by k-means clustering, with the number of clusters ranging from 1 to 30 (just 1 cluster is identical to single-matrix LMNN). As k-means clustering is prone to local minima, we averaged these results over 100 runs. The figure shows the average test error rates in red, as well as their standard deviations (via error bars). In the supervised setting, the training examples were partitioned by their class labels, resulting in the same number of clusters as classes. The test error rates in these experiments are shown as blue crosses. In both the unsupervised and supervised settings, the test error rates decreased with the use of multiple metrics. However, the improvements were far greater in the supervised setting.

[1] http://yann.lecun.com/exdb/mnist/
[2] http://people.csail.mit.edu/jrennie/20Newsgroups
[3] http://www.ics.uci.edu/~mlearn/databases/letter-recognition/letter-recognition.names
[4] http://archive.ics.uci.edu/ml/
[5] http://cvc.yale.edu/projects/yalefacesB/yalefacesB.html

	Error in %	mnist	20news	letters	isolet	yalefaces
train	LMNN	1.72	14.91	3.62	3.59	**6.48**
	Multiple Metrics	**1.18**	**13.66**	**3.2**	**3.08**	6.4
test	LMNN	1.19	9.73	3.54	0.7	**3.54**
	Multiple Metrics	**0.04**	**7.08**	**1.55**	**0**	3.57

Figure 5. The classification train- and testerror rates with one metric (LMNN) and multiple metrics. The value of k was set by cross validation.

Finally, our last experiments explored the improvement in kNN error rates when one distance metric was learned for the training examples in each class. In these experiments, we used the full number of training examples for each data set. In addition, we used PCA to project the training inputs into a lower dimensional subspace accounting for at least 95% of the data's total variance. Fig. 5 shows generally consistent improvement in training and test kNN error rates, though overfitting is an issue, especially on the 20-NewsGroups and YaleFaces data sets. This overfitting is to be expected from the relatively large number of classes and high input dimensionality of these data sets: the number of model parameters in these experiments grows linearly in the former and quadratically in the latter. On these data sets, only the use of a validation set prevents the training error from vanishing completely while the test error skyrockets. On the other hand, a significant improvement in the test error rate is observed on the largest data set, that of MNIST handwritten digits. On this data set, multiple distance metrics yield a 1.18% test error rate—a highly competitive result for a method that does not take into account prior domain knowledge (LeCun et al., 1998).

6. Discussion

In this paper, we have extended the original framework for LMNN classification in several important ways: by describing a solver that scales well to larger data sets, by integrating metric ball trees into the training and testing procedures, by exploring the use of dimensionality reduction for further speedups, and by showing how to train different Mahalanobis distance metrics in different parts of the input space. These extensions should prove useful in many applications of kNN classification. More generally, we also hope they spur further work on problems in distance metric learning and large-scale semidefinite programming, both areas of great interest in the larger field of machine learning.

Acknowledgments

This research is based upon work supported by the National Science Foundation under Grant No. 0238323.

References

Borchers, B. (1999). CSDP, a C library for semidefinite programming. *Optimization Methods and Software 11(1):613-623*.

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

Chopra, S., Hadsell, R., & LeCun, Y. (2005). Learning a similiarty metric discriminatively, with application to face verification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-05)*. San Diego, CA.

Davis, J., Kulis, B., Jain, P., Sra, S., & Dhillon, I. (2007). Information-theoretic metric learning. *Proceedings of the 24th International Conference on Machine Learning (ICML-07)*. Corvallis, OR.

Frome, A., Singer, Y., Sha, F., & Malik, J. (2007). Learning globally-consistent local distance functions for shape-based image retrieval and classification. *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV-07)*. Rio de Janeiro, Brazil.

Goldberger, J., Roweis, S., Hinton, G., & Salakhutdinov, R. (2005). Neighbourhood components analysis. *Advances in Neural Information Processing Systems 17* (pp. 513–520). Cambridge, MA: MIT Press.

Hastie, T., & Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *18*, 607–616.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86(11)*, 2278–2324.

Liu, T., Moore, A. W., Gray, A., & Yang, K. (2005). An investigation of practical approximate nearest neighbor algorithms. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*, 825–832. Cambridge, MA: MIT Press.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. Cambridge, MA: MIT Press.

Simard, P. Y., LeCun, Y., & Decker, J. (1993). Efficient pattern recognition using a new transformation distance. *Advances in Neural Information Processing Systems* (pp. 50–58). San Mateo, CA: Morgan Kaufman.

Torresani, L., & Lee, K. (2007). Large margin component analysis. In B. Schölkopf, J. Platt and T. Hofmann (Eds.), *Advances in neural information processing systems 19*. Cambridge, MA: MIT Press.

Weinberger, K., Blitzer, J., & Saul, L. (2006). Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf and J. Platt (Eds.), *Advances in neural information processing systems 18*. Cambridge, MA: MIT Press.

Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2002). Distance metric learning, with application to clustering with side-information. *Advances in Neural Information Processing Systems 14*. Cambridge, MA: MIT Press.