Journal of Machine Learning Research 10 (2009) 207-244 Submitted 12/07; Revised 9/08; Published 2/09

Distance Metric Learning for Large Margin Nearest Neighbor Classification

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

Lawrence K. Saul SAUL@CS.UCSD.EDU
Department of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, Mail Code 0404
La Jolla, CA 92093-0404

Editor: Sam Roweis

Abstract

The accuracy of *k*-nearest neighbor (kNN) classification depends significantly on the metric used to compute distances between different examples. In this paper, we show how to learn a Mahalanobis distance metric for kNN classification from labeled examples. The Mahalanobis metric can equivalently be viewed as a global linear transformation of the input space that precedes kNN classification using Euclidean distances. In our approach, the metric is trained with the goal that the *k*-nearest neighbors always belong to the same class while examples from different classes are separated by a large margin. As in support vector machines (SVMs), the margin criterion leads to a convex optimization based on the hinge loss. Unlike learning in SVMs, however, our approach requires no modification or extension for problems in multiway (as opposed to binary) classification. In our framework, the Mahalanobis distance metric is obtained as the solution to a semidefinite program. On several data sets of varying size and difficulty, we find that metrics trained in this way lead to significant improvements in kNN classification. Sometimes these results can be further improved by clustering the training examples and learning an individual metric within each cluster. We show how to learn and combine these local metrics in a globally integrated manner.

Keywords: convex optimization, semi-definite programming, Mahalanobis distance, metric learning, multi-class classification, support vector machines

1. Introduction

One of the oldest and simplest methods for pattern classification is the *k*-nearest neighbors (kNN) rule (Cover and Hart, 1967). The kNN rule classifies each unlabeled example by the majority label of its *k*-nearest neighbors in the training set. Despite its simplicity, the kNN rule often yields competitive results and in certain domains, when cleverly combined with prior knowledge, it has significantly advanced the state-of-the-art (Belongie et al., 2002; Simard et al., 1993).

By the very nature of its decision rule, the performance of kNN classification depends crucially on the way that distances are computed between different examples. When no prior knowledge is available, most implementations of kNN compute simple Euclidean distances (assuming the examples are represented as vector inputs). Unfortunately, Euclidean distances ignore any statistical

regularities that might be estimated from a large training set of labeled examples. Ideally, one would like to adapt the distance metric to the application at hand. Suppose, for example, that we are using kNN to classify images of faces by age and gender. It can hardly be optimal to use the same distance metric for age and gender classification, even if in both tasks, distances are computed between the same sets of extracted features (e.g., pixels, color histograms).

Motivated by these issues, a number of researchers have demonstrated that kNN classification can be greatly improved by learning an appropriate distance metric from labeled examples (Chopra et al., 2005; Goldberger et al., 2005; Shalev-Shwartz et al., 2004; Shental et al., 2002). This is the so-called problem of *distance metric learning*. Recently, it has been shown that even a simple linear transformation of the input features can lead to significant improvements in kNN classification (Goldberger et al., 2005; Shalev-Shwartz et al., 2004). Our work builds in a novel direction on the success of these previous approaches.

In this paper, we show how to learn a Mahalanobis distance metric for kNN classification. The algorithm that we propose was described at a high level in earlier work (Weinberger et al., 2006) and later extended in terms of scalability and accuracy (Weinberger and Saul, 2008). Intuitively, the algorithm is based on the simple observation that the kNN decision rule will correctly classify an example if its k-nearest neighbors share the same label. The algorithm attempts to increase the number of training examples with this property by learning a linear transformation of the input space that precedes kNN classification using Euclidean distances. The linear transformation is derived by minimizing a loss function that consists of two terms. The first term penalizes large distances between examples in the same class that are desired as k-nearest neighbors, while the second term penalizes small distances between examples with non-matching labels. Minimizing these terms yields a linear transformation of the input space that increases the number of training examples whose k-nearest neighbors have matching labels. The Euclidean distances in the transformed space can equivalently be viewed as Mahalanobis distances in the original space. We exploit this equivalence to cast the problem of distance metric learning as a problem in convex optimization.

Our approach is largely inspired by recent work on neighborhood component analysis (Goldberger et al., 2005) and metric learning in energy-based models (Chopra et al., 2005). Despite similar goals, however, our method differs significantly in the proposed optimization. We formulate the problem of distance metric learning as an instance of semidefinite programming. Thus, the optimization is convex, and its global minimum can be efficiently computed. There have been other studies in distance metric learning based on eigenvalue problems (Shental et al., 2002; De Bie et al., 2003) and semidefinite programming (Globerson and Roweis, 2006; Shalev-Shwartz et al., 2004; Xing et al., 2002). These previous approaches, however, essentially attempt to learn distance metrics that cluster together *all* similarly labeled inputs, even those that are not k-nearest neighbors. This objective is far more difficult to achieve than what we propose. Moreover, it does not leverage the full power of kNN classification, whose accuracy does not require that all similarly labeled inputs be tightly clustered.

There are many parallels between our method and classification by support vector machines (SVMs)—most notably, a convex objective function based on the hinge loss, and the potential to work in nonlinear feature spaces by using the "kernel trick". In light of these parallels, we describe our approach as *large margin nearest neighbor* (LMNN) classification. Our framework can be viewed as the logical counterpart to SVMs in which kNN classification replaces linear classification.

Our framework contrasts with classification by SVMs, however, in one intriguing respect: it requires no modification for multiclass problems. Extensions of SVMs to multiclass problems typi-

cally involve combining the results of many binary classifiers, or they require additional machinery that is elegant but non-trivial (Crammer and Singer, 2001). In both cases the training time scales at least linearly in the number of classes. By contrast, our framework has no explicit dependence on the number of classes.

We also show how to extend our framework to learn multiple Mahalanobis metrics, each of them associated with a different class label and/or region of the input space. The multiple metrics are trained simultaneously by minimizing a single loss function. While the loss function couples metrics in different parts of the input space, the optimization remains an instance of semidefinite programming. The globally integrated training of local distance metrics distinguishes our approach from earlier work on discriminant adaptive kNN classification (Hastie and Tibshirani, 1996)

Our paper is organized as follows. Section 2 introduces the general problem of distance metric learning for *k*NN classification and reviews previous approaches that motivated our work. Section 3 describes our model for LMNN classification and formulates the required optimization as an instance of semidefinite programming. Section 4 presents experimental results on several data sets. Section 5 discusses several extensions to LMNN classification, including iterative re-estimation of target neighbors, locally adaptive Mahalanobis metrics in different parts of the input space, and "kernelization" of the basic algorithm. Section 6 describes faster implementations for training and testing in LMNN classification using ball trees. Section 7 concludes by summarizing our main contributions and sketching several directions of ongoing research. Finally, appendix A describes the special-purpose solver that we implemented for large scale problems in LMNN classification.

2. Background

In this section, we introduce the general problem of distance metric learning (section 2.1) and review a number of previously studied approaches. Broadly speaking, these approaches fall into three categories: eigenvector methods based on second-order statistics (section 2.2), convex optimizations over the space of positive semidefinite matrices (section 2.3), and fully supervised algorithms that directly attempt to optimize *k*NN classification error (section 2.4) .

2.1 Distance Metric Learning

We begin by reviewing some basic terminology. A mapping $D : \mathcal{X} \times \mathcal{X} \to \Re_0^+$ over a vector space $\mathcal{X}$ is called a **metric** if for all vectors $\forall \vec{x}_i, \vec{x}_j, \vec{x}_k \in \mathcal{X}$, it satisfies the properties:

1. $D(\vec{x}_i, \vec{x}_j) + D(\vec{x}_j, \vec{x}_k) \;\geq\; D(\vec{x}_i, \vec{x}_k)$ (triangular inequality).
2. $D(\vec{x}_i, \vec{x}_j) \geq 0$ (non-negativity).
3. $D(\vec{x}_i, \vec{x}_j) = D(\vec{x}_j, \vec{x}_i)$ (symmetry).
4. $D(\vec{x}_i, \vec{x}_j) = 0 \iff \vec{x}_i = \vec{x}_j$ (distinguishability).

Strictly speaking, if a mapping satisfies the first three properties but not the fourth, it is called a **pseudometric**. However, to simplify the discussion in what follows, we will often refer to pseudometrics as metrics, pointing out the distinction only when necessary.

We obtain a family of metrics over $\mathcal{X}$ by computing Euclidean distances after performing a linear transformation $\vec{x}' = \mathbf{L}\vec{x}$. These metrics compute squared distances as:

$$D_{\mathbf{L}}(\vec{x}_i, \vec{x}_j) = \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|_2^2, \tag{1}$$

where the linear transformation in Eq. (1) is parameterized by the matrix $\mathbf{L}$. It is simple to show that Eq. (1) defines a valid metric if $\mathbf{L}$ is full rank and a valid pseudometric otherwise.

It is common to express squared distances under the metric in Eq. (1) in terms of the square matrix:

$$\mathbf{M} = \mathbf{L}^\top \mathbf{L}. \tag{2}$$

Any matrix $\mathbf{M}$ formed in this way from a real-valued matrix $\mathbf{L}$ is guaranteed to be positive semidefinite (i.e., to have no negative eigenvalues). In terms of the matrix $\mathbf{M}$, we denote squared distances by

$$\mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j), \tag{3}$$

and we refer to pseudometrics of this form as **Mahalanobis** metrics. Originally, this term was used to describe the quadratic forms in Gaussian distributions, where the matrix $\mathbf{M}$ played the role of the inverse covariance matrix. Here we allow $\mathbf{M}$ to denote any positive semidefinite matrix. The distances in Eq. (1) and Eq. (3) can be viewed as generalizations of Euclidean distances. In particular, Euclidean distances are recovered by setting $\mathbf{M}$ to be equal to the identity matrix.

A Mahalanobis distance metric can be parameterized in terms of the matrix $\mathbf{L}$ or the matrix $\mathbf{M}$. Note that the matrix $\mathbf{L}$ uniquely defines the matrix $\mathbf{M}$, while the matrix $\mathbf{M}$ defines $\mathbf{L}$ up to rotation (which does not affect the computation of distances). This equivalence suggests two different approaches to distance metric learning. In particular, we can either estimate a linear transformation $\mathbf{L}$, or we can estimate a positive semidefinite matrix $\mathbf{M}$. Note that in the first approach, the optimization is unconstrained, while in the second approach, it is important to enforce the constraint that the matrix $\mathbf{M}$ is positive semidefinite. Though generally more complicated to solve a constrained optimization, this second approach has certain advantages that we explore in later sections.

Many researchers have proposed ways to estimate Mahalanobis distance metrics for the purpose of computing distances in kNN classification. In particular, let $\{(\vec{x}_i, y_i)\}_{i=1}^n$ denote a training set of n labeled examples with inputs $\vec{x}_i \in \Re^d$ and discrete (but not necessarily binary) class labels $y_i \in \{1, 2, \ldots, C\}$. For kNN classification, one seeks a linear transformation such that nearest neighbors computed from the distances in Eq. (1) share the same class labels. We review several previous approaches to this problem in the following section.

2.2 Eigenvector Methods

Eigenvector methods have been widely used to discover informative linear transformations of the input space. As discussed in section 2.1, these linear transformations can be viewed as inducing a Mahalanobis distance metric. Popular eigenvector methods for linear preprocessing are principal component analysis, linear discriminant analysis, and relevant component analysis. These methods differ in the way that they use labeled or unlabeled data to derive linear transformations of the input space. These methods can also be "kernelized" to work in a nonlinear feature space (Müller et al., 2001; Schölkopf et al., 1998; Tsang et al., 2005), though we do not discuss such formulations here.

2.2.1 PRINCIPAL COMPONENT ANALYSIS

We briefly review principal component analysis (PCA) (Jolliffe, 1986) in the context of distance metric learning. Essentially, PCA computes the linear transformation $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ that projects the training inputs $\{\vec{x}_i\}_{i=1}^n$ into a variance-maximizing subspace. The variance of the projected inputs

can be written in terms of the covariance matrix:

$$\mathbf{C} = \frac{1}{n}\sum_{i=1}^{n}(\vec{x}_i - \vec{\mu})^\top(\vec{x}_i - \vec{\mu}),$$

where $\vec{\mu} = \frac{1}{n}\sum_i \vec{x}_i$ denotes the sample mean. The linear transformation $\mathbf{L}$ is chosen to maximize the variance of the projected inputs, subject to the constraint that $\mathbf{L}$ defines a projection matrix. In terms of the input covariance matrix, the required optimization is given by:

$$\max_{\mathbf{L}} \mathrm{Tr}(\mathbf{L}^\top \mathbf{C}\mathbf{L}) \text{ subject to: } \mathbf{L}\mathbf{L}^\top = \mathbf{I}. \tag{4}$$

The optimization in Eq. (4) has a closed-form solution; the standard convention equates the rows of $\mathbf{L}$ with the leading eigenvectors of the covariance matrix. If $\mathbf{L}$ is a rectangular matrix, the linear transformation projects the inputs into a lower dimensional subspace. If $\mathbf{L}$ is a square matrix, then the transformation does not reduce the dimensionality, but this solution still serves to rotate and re-order the input coordinates by their respective variances.

Note that PCA operates in an unsupervised setting without using the class labels of training inputs to derive informative linear projections. Nevertheless, PCA still has certain useful properties as a form of linear preprocessing for kNN classification. For example, PCA can be used for "denoising": projecting out the components of the bottom eigenvectors often reduces kNN error rate. PCA can also be used to accelerate neighbor nearest computations in large data sets. The linear preprocessing from PCA can significantly reduce the amount of computation either by explicitly reducing the dimensionality of the inputs, or simply by re-ordering the input coordinates in terms of their variance (as discussed further in section 6).

2.2.2 LINEAR DISCRIMINANT ANALYSIS

We briefly review linear discriminant analysis (LDA) (Fisher, 1936) in the context of distance metric learning. Let Ω_c denote the set of indices of examples in the cth class (with $y_i = c$). Essentially, LDA computes the linear projection $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ that maximizes the amount of between-class variance relative to the amount of within-class variance. These variances are computed from the between-class and within-class covariance matrices, defined by:

$$\begin{aligned} \mathbf{C}_b &= \frac{1}{C}\sum_{c=1}^{C}\vec{\mu}_c\vec{\mu}_c^\top, \\ \mathbf{C}_w &= \frac{1}{n}\sum_{c=1}^{C}\sum_{i\in\Omega_c}(\vec{x}_i - \vec{\mu}_c)(\vec{x}_i - \vec{\mu}_c)^\top, \end{aligned} \tag{5}$$

where $\vec{\mu}_c$ denotes the sample mean of the c^{th} class; we also assume that the data is globally centered. The linear transformation $\mathbf{L}$ is chosen to maximize the ratio of between-class to within-class variance, subject to the constraint that $\mathbf{L}$ defines a projection matrix. In terms of the above covariance matrices, the required optimization is given by:

$$\max_{\mathbf{L}} \mathrm{Tr}\left(\frac{\mathbf{L}^\top\mathbf{C}_b\mathbf{L}}{\mathbf{L}^\top\mathbf{C}_w\mathbf{L}}\right) \text{ subject to: } \mathbf{L}\mathbf{L}^\top = \mathbf{I}. \tag{6}$$

The optimization in Eq. (6) has a closed-form solution; the standard convention equates the rows of $\mathbf{L}$ with the leading eigenvectors of $\mathbf{C}_w^{-1}\mathbf{C}_b$.

LDA is widely used as a form of linear preprocessing for pattern classification. Unlike PCA, LDA operates in a supervised setting and uses the class labels of the inputs to derive informative linear projections. Note that the between-class covariance matrix $\mathbf{C}_b$ in Eq. (5) has at most rank $\mathcal{C}$, where $\mathcal{C}$ is the number of classes. Thus, up to $\mathcal{C}$ linear projections can be extracted from the eigenvalue problem in LDA. Because these projections are based on second-order statistics, they work well to separate classes whose conditional densities are multivariate Gaussian. When this assumption does not hold, however, LDA may extract spurious features that are not well suited to kNN classification.

2.2.3 RELEVANT COMPONENT ANALYSIS

Finally, we briefly review relevant component analysis (RCA) (Shental et al., 2002; Bar-Hillel et al., 2006) in the context of distance metric learning. RCA is intermediate between PCA and LDA in its use of labeled data. Specifically, RCA makes use of so-called "chunklet" information, or subclass membership assignments. A chunklet is essentially a subset of a class. Inputs in the same chunklet belong to the same class, but inputs in different chunklets do not necessarily belong to different classes. Essentially, RCA computes the linear projection $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ that "whitens" the data with respect to the averaged within-chunklet covariance matrix. In particular, let Ω_ℓ denote the set of indices of examples in the ℓth chunklet, and let $\vec{\mu}_\ell$ denote the mean of these examples. The averaged within-chunklet covariance matrix is given by:

$$\mathbf{C}_w = \frac{1}{n}\sum_{l=1}^{L}\sum_{i\in\Omega_l}(\vec{x}_i - \vec{\mu}_l)(\vec{x}_i - \vec{\mu}_l)^\top.$$

RCA uses the linear transformation $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ with $\mathbf{L} = \mathbf{C}_w^{-1/2}$. This transformation acts to normalize the within-chunklet variance. An unintended side effect of this transformation may be to amplify noisy directions in the data. Thus, it is recommended to de-noise the data by PCA before computing the within-chunklet covariance matrix.

2.3 Convex Optimization

Recall that the goal of distance metric learning can be stated in two ways: to learn a linear transformation $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ or, equivalently, to learn a Mahalanobis metric $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$. It is possible to formulate certain types of distance metric learning as convex optimizations over the cone of positive semidefinite matrices $\mathbf{M}$. In this section, we review two previous approaches based on this idea.

2.3.1 MAHALANOBIS METRIC FOR CLUSTERING

A convex objective function for distance metric learning was first proposed by Xing et al. (2002). The goal of this work was to learn a Mahalanobis metric for clustering (MMC) with side-information. MMC shares a similar goal as LDA: namely, to minimize the distances between similarly labeled inputs while maximizing the distances between differently labeled inputs. MMC differs from LDA in its formulation of distance metric learning as an convex optimization problem. In particular, whereas LDA solves the eigenvalue problem in Eq. (6) to compute the linear transformation $\mathbf{L}$, MMC solves a convex optimization over the matrix $\mathbf{M} = \mathbf{L}^\top\mathbf{L}$ that directly represents the Mahalanobix metric itself.

To state the optimization for MMC, it is helpful to introduce further notation. From the class labels y_i, we define the $n \times n$ binary association matrix with elements $y_{ij} = 1$ if $y_i = y_j$ and $y_{ij} = 0$ otherwise. In terms of this notation, MMC attempts to maximize the distances between pairs of inputs with different labels ($y_{ij} = 0$), while constraining the sum over squared distances of pairs of similarly labeled inputs ($y_{ij} = 1$). In particular, MMC solves the following optimization:

$$\begin{aligned} &\textbf{Maximize } \textstyle\sum_{ij}(1-y_{ij})\sqrt{\mathcal{D}_{\mathbf{M}}(\vec{x}_i,\vec{x}_j)} \textbf{ subject to:} \\ &\qquad \textbf{(1)} \textstyle\sum_{ij} y_{ij}\mathcal{D}_{\mathbf{M}}(\vec{x}_i,\vec{x}_j) \;\leq\; 1 \\ &\qquad \textbf{(2)} \; \mathbf{M} \succeq 0\textbf{.} \end{aligned}$$

The first constraint is required to make the problem feasible and bounded; the second constraint enforces that $\mathbf{M}$ is a positive semidefinite matrix. The overall optimization is convex. The square root in the objective function ensures that MMC leads to generally different results than LDA.

MMC was designed to improve the performance of iterative clustering algorithms such as *k*-means. In these algorithms, clusters are generally modeled as normal or unimodal distributions. MMC builds on this assumption by attempting to minimize distances between all pairs of similarly labeled inputs; this objective is only sensible for unimodal clusters. For this reason, however, MMC is not especially appropriate as a form of distance metric learning for *k*NN classification. One of the major strengths of *k*NN classification is its non-parametric framework. Thus a different objective for distance metric learning is needed to preserve this strength of *k*NN classification—namely, that it does not implicitly make parametric (or other limiting) assumptions about the input distributions.

2.3.2 ONLINE LEARNING OF MAHALANOBIS DISTANCES

Convex optimizations over the cone of positive semidefinite matrices have also been proposed for perceptron-like approaches to distance metric learning. The Pseudometric Online Learning Algorithm (POLA) (Shalev-Shwartz et al., 2004) combines ideas from convex optimization and large margin classification. Like LDA and MMC, POLA attempts to learn a metric that shrinks distances between similarly labeled inputs and expands distances between differently labeled inputs. POLA differs from LDA and MMC, however, in explicitly encouraging a finite margin that separates differently labeled inputs. POLA was also conceived in an online setting.

The online version of POLA works as follows. At time t, the learning environment presents a tuple $(\vec{x}_t, \vec{x}'_t, y_t)$, where the binary label y_t indicates whether the two inputs $\vec{x}_t$ and $\vec{x}'_t$ belong to the same ($y_t = 1$) or different ($y_t = -1$) classes. From streaming tuples of this form, POLA attempts to learn a Mahalanobis metric $\mathbf{M}$ and a scalar threshold b such that similarly labeled inputs are *at most* a distance of $b-1$ apart, while differently labeled inputs are *at least* a distance of $b+1$ apart. These constraints can be expressed by the single inequality:

$$y_t \left[b - \left(\vec{x}_t - \vec{x}'_t\right)^\top \mathbf{M} \left(\vec{x}_t - \vec{x}'_t\right) \right] \geq 1. \tag{7}$$

The distance metric $\mathbf{M}$ and threshold b are updated after each tuple $(\vec{u}_t, \vec{v}_t, y_t)$ to correct any violation of this inequality. In particular, the update computes a positive semidefinite matrix $\mathbf{M}$ that satisfies (7). The required optimization can be performed by an alternating projection algorithm, similar to the one described in appendix A. The algorithm extends naturally to problems with more than two classes.

POLA can also be implemented on a data set of fixed size. In this setting, pairs of inputs are repeatedly processed until no pair violates its margin constraints by more than some constant $\beta > 0$. Moreover, as in perceptron learning, the number of iterations over the data set can be bounded above (Shalev-Shwartz et al., 2004).

In many ways, POLA exhibits the same strengths and weaknesses as MMC. Both algorithms are based on convex optimizations that do not have spurious local minima. On the other hand, both algorithms make implicit assumptions about the distributions of inputs and class labels. The margin constraints enforced by POLA are designed to learn a distance metric under which all pairs of similarly labeled inputs are closer than all pairs of differently labeled inputs. This type of learning may often be unrealizable, however, even in situations where *k*NN classification is able to succeed. For this reason, a different framework is required to learn distance metrics for *k*NN classification.

2.4 Neighborhood Component Analysis

Recently, Goldberger et al. (2005) considered how to learn a Mahalnobis distance metric especially for *k*NN classification. They proposed a novel supervised learning algorithm known as *Neighborhood Component Analysis* (NCA). The algorithm computes the expected leave-one-out classification error from a stochastic variant of *k*NN classification. The stochastic classifier uses a Mahalanobis distance metric parameterized by the linear transformation $\vec{x} \to \mathbf{L}\vec{x}$ in Eqs. (1–3). The algorithm attempts to estimate the linear transformation $\mathbf{L}$ that minimizes the expected classification error when distances are computed in this way.

The stochastic classifier in NCA is used to label queries by the majority vote of nearby training examples, but not necessarily the k nearest neighbors. In particular, for each query, the reference examples in the training set are drawn from a softmax probability distribution that favors nearby examples over faraway ones. The probability of drawing $\vec{x}_j$ as a reference example for $\vec{x}_i$ is given by:

$$p_{ij} = \begin{cases} \frac{\exp\left(-\|\mathbf{L}x_i - \mathbf{L}x_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|\mathbf{L}x_i - \mathbf{L}x_k\|^2\right)} & \text{if } i \neq j \\ 0 & \text{if } i = j. \end{cases} \tag{8}$$

Note that there is no free parameter k for the number of nearest neighbors in this stochastic classifier. Instead, the scale of $\mathbf{L}$ determines the size of neighborhoods from which nearby training examples are sampled. On average, though, this sampling procedure yields similar results as a deterministic *k*NN classifier (for some value of k) with the same Mahalanobis distance metric.

Under the softmax sampling scheme in Eq. (8), it is simple to compute the expected leave-one-out classification error on the training examples. As in section 2.3.1, we define the $n \times n$ binary matrix with elements $y_{ij} = 1$ if $y_i = y_j$ and $y_{ij} = 0$ otherwise. The expected error computes the fraction of training examples that are (on average) misclassified:

$$\varepsilon_{\text{NCA}} = 1 - \frac{1}{n} \sum_{ij} p_{ij} y_{ij}. \tag{9}$$

The error in Eq. (9) is a continuous, differentiable function of the linear transformation $\mathbf{L}$ used to compute Mahalanobis distances in Eq. (8).

Note that the differentiability of Eq. (9) depends on the stochastic neighborhood assignment of the NCA decision rule. By contrast, the leave-one-out error of a deterministic *k*NN classifier is neither continuous nor differentiable in the parameters of the distance metric. For distance metric

learning, the differentiability of Eq. (9) is a key advantage of stochastic neighborhood assignment, making it possible to minimize this error measure by gradient descent. It would be much more difficult to minimize the leave-one-out error of its deterministic counterpart.

The objective function for NCA differs in one important respect from other algorithms reviewed in this section. Though continuous and differentiable with respect to the parameters of the distance metric, Eq. (9) is not convex, nor can it be minimized using eigenvector methods. Thus, the optimization in NCA can suffer from spurious local minima. In practice, the results of the learning algorithm depend on the initialization of the distance metric.

The linear transformation in NCA can also be used to project the inputs into a lower dimensional Euclidean space. Eqs. (8–9) remain valid when $\mathbf{L}$ is a rectangular as opposed to square matrix. Lower dimensional projections learned by NCA can be used to visualize class structure and/or to accelerate kNN search.

Recently, Globerson and Roweis (2006) proposed a related model known as Metric Learning by Collapsing Classes (MLCC). The goal of MLCC is to find a distance metric that (like LDA) shrinks the within-class variance while maintaining the separation between different classes. MLCC uses a similar rule as NCA for stochastic classification, so as to yield a differentiable objective function. Compared to NCA, MLCC has both advantages and disadvantages for distance metric learning. The main advantage is that distance metric learning in MLCC can be formulated as a convex optimization over the space of positive semidefinite matrices. The main disadvantage is that MLCC implicitly assumes that the examples in each class have a unimodal distribution. In this sense, MLCC shares the same basic strengths and weaknesses of the methods described in section 2.3.

3. Model

The model we propose for distance metric learning builds on the algorithms reviewed in section 2. In common with all of them, we attempt to learn a Mahalanobis distance metric of the form in Eqs. (1–3). Other key aspects of our model build on the particular strengths of individual approaches. As in MMC (see section 2.3.1), we formulate the parameter estimation in our model as a convex optimization over the space of positive semidefinite matrices. As in POLA (see section 2.3.2), we attempt to maximize the margin by which the model correctly classifies labeled examples in the training set. Finally, as in NCA (see section 2.4), our model was conceived specifically to learn a Mahalanobis distance metric that improves the accuracy of kNN classification. Indeed, the three essential ingredients of our model are (i) its convex loss function, (ii) its goal of margin maximization, and (iii) the constraints on the distance metric imposed by accurate kNN classification.

3.1 Intuition and Terminology

Our model is based on two simple intuitions (and idealizations) for robust kNN classification: first, that each training input $\vec{x}_i$ should share the same label y_i as its k nearest neighbors; second, that training inputs with different labels should be widely separated. We attempt to learn a linear transformation of the input space such that the training inputs satisfy these properties. In fact, these objectives are neatly balanced by two competing terms in our model's loss function. Specifically, one term penalizes large distances between nearby inputs with the same label, while the other term

penalizes small distances between inputs with different labels. To make precise these relative notions of "large" and "small", however, we first need to introduce some new terminology.

Learning in our framework requires auxiliary information beyond the label y_i of each input $\vec{x}_i$ in the training set. Recall that the goal of learning is to estimate a distance metric under which each input $\vec{x}_i$ has k nearest neighbors that share its same label y_i. We facilitate this goal by identifying *target neighbors* for each input $\vec{x}_i$ at the outset of learning. The target neighbors of $\vec{x}_i$ are those that we desire to be closest to $\vec{x}_i$; in particular, we attempt to learn a linear transformation of the input space such that the resulting nearest neighbors of $\vec{x}_i$ are indeed its target neighbors. We emphasize that target neighbors are fixed a priori and do not change during the learning process. This step significantly simplifies the learning process by specifying a priori which similarly labeled inputs to cluster together. In many applications, there may be prior knowledge or auxiliary information (e.g., a similarity graph) that naturally identifies target neighbors. In the absence of prior knowledge, the simplest prescription is to compute the k nearest neighbors with the same class label, as determined by Euclidean distance. This was done for all the experiments in this paper. We use the notation $j \rightsquigarrow i$ to indicate that input $\vec{x}_j$ is a target neighbor of input $\vec{x}_i$. Note that this relation is not symmetric: $j \rightsquigarrow i$ does not imply $i \rightsquigarrow j$.

For kNN classification to succeed, the target neighbors of each input $\vec{x}_i$ should be closer than all differently labeled inputs. In particular, for each input $\vec{x}_i$, we can imagine the target neighbors as establishing a perimeter that differently labeled inputs should not invade. We refer to the differently labeled inputs in the training set that invade this perimeter as *impostors*; the goal of learning (roughly speaking) is to minimize the number of impostors.

In fact, to increase the robustness of kNN classification, we adopt an even more stringent goal for learning—namely to maintain a large (finite) distance between impostors and the perimeters established by target neighbors. By maintaining a *margin* of safety around the kNN decision boundaries, we ensure that the model is robust to small amounts of noise in the training inputs. This robustness criterion also gives rise to the name of our approach: *large margin nearest neighbor* (LMNN) classification.

In mathematical terms, impostors are defined by a simple inequality. For an input $\vec{x}_i$ with label y_i and target neighbor $\vec{x}_j$, an impostor is any input $\vec{x}_l$ with label $\vec{y}_l \neq \vec{y}_i$ such that

$$\|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2 \leq \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 + 1. \tag{10}$$

In other words, an impostor $\vec{x}_l$ is any differently labeled input that invades the perimeter plus unit margin defined by any target neighbor $\vec{x}_j$ of the input $\vec{x}_i$.

Figure 1 illustrates the main idea behind LMNN classification. Before learning, a training input has both target neighbors and impostors in its local neighborhood. During learning, the impostors are pushed outside the perimeter established by the target neighbors. After learning, there exists a finite margin between the perimeter and the impostors. The figure shows the idealized scenario where kNN classification errors in the original input space are corrected by learning an appropriate linear transformation.

3.2 Loss Function

With the intuition and terminology from the previous section, we can now construct a loss function for LMNN classification. The loss function consists of two terms, one which acts to *pull* target neighbors closer together, and another which acts to *push* differently labeled examples further apart.

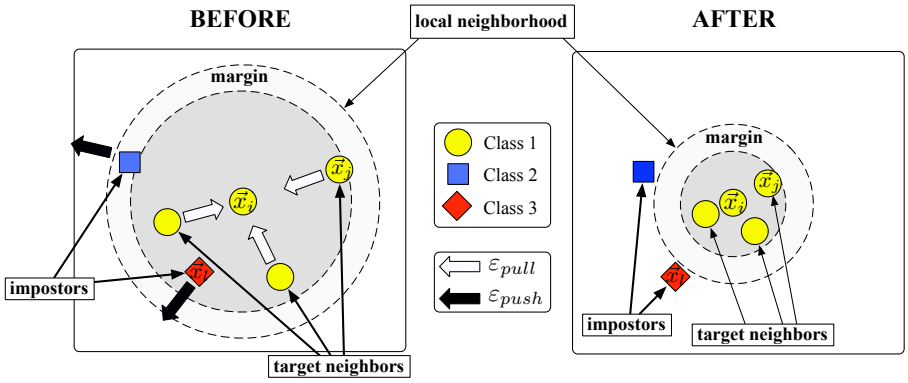

Figure 1: Schematic illustration of one input's neighborhood before training (*left*) versus after training (*right*). The distance metric is optimized so that: (i) its $k=3$ target neighbors lie within a smaller radius after training; (ii) differently labeled inputs lie outside this smaller radius by some finite margin. Arrows indicate the gradients on distances arising from different terms in the cost function.

These two terms have competing effects, since the first is reduced by shrinking the distances between examples while the second is generally reduced by magnifying them. We discuss each term in turn.

The first term in the loss function penalizes large distances between each input and its target neighbors. In terms of the linear transformation $\mathbf{L}$ of the input space, the sum of these squared distances is given by:

$$\varepsilon_{\text{pull}}(\mathbf{L}) = \sum_{j \rightsquigarrow i} \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2. \tag{11}$$

The gradient of this term generates a pulling force that attracts target neighbors in the linearly transformed input space. It is important that Eq. (11) only penalizes large distances between inputs and their *target neighbors*; in particular, it does not penalize large distances between all similarly labeled inputs. We purposefully do not penalize the latter because accurate kNN classification does not require that all similarly labeled inputs be tightly clustered. Our approach is distinguished in this way from many previous approaches to distance metric learning; see section 2. By only penalizing large distances between neighbors, we build models that leverage the full power of kNN classification.

The second term in the loss function penalizes small distances between differently labeled examples. In particular, the term penalizes violations of the inequality in Eq. (10). To simplify notation, we introduce a new indicator variable $y_{il} = 1$ if and only if $y_i = y_l$, and $y_{il} = 0$ otherwise. In terms of this notation, the second term of the loss function ε_{push} is given by:

$$\varepsilon_{\text{push}}(\mathbf{L}) = \sum_{i,j \rightsquigarrow i} \sum_{l} (1 - y_{il}) \left[1 + \|\mathbf{L}(\vec{x}_i - \vec{x}_j)\|^2 - \|\mathbf{L}(\vec{x}_i - \vec{x}_l)\|^2\right]_+ \tag{12}$$

where the term $[z]_+ = \max(z, 0)$ denotes the standard hinge loss. The hinge loss monitors the inequality in Eq. (10). If the inequality does not hold (i.e., the input $\vec{x}_l$ lies a safe distance away from $\vec{x}_i$), then its hinge loss has a negative argument and makes no contribution to the overall loss func-

tion. The (sub-)gradient of Eq. (12) generates a pushing force that repels imposters away from the perimeter established by each example's k nearest (similarly labeled) neighbors; see Fig. 1.

The choice of unit margin is an arbitrary convention that sets the scale for the linear transformation $\mathbf{L}$ (which enters every other term in the loss function). If a margin $c > 0$ was enforced instead of the unit margin, the loss function would be minimized by the same linear transformation up to an overall scale factor $\sqrt{c}$.

Finally, we combine the two terms $\varepsilon_{\text{pull}}(\mathbf{L})$ and $\varepsilon_{\text{push}}(\mathbf{L})$ into a single loss function for distance metric learning. The two terms can have competing effects—to attract target neighbors on one hand, to repel impostors on the other. A weighting parameter $\mu \in [0,1]$ balances these goals:

$$\varepsilon(\mathbf{L}) = (1-\mu)\,\varepsilon_{pull}(\mathbf{L}) + \mu\varepsilon_{push}(\mathbf{L}). \tag{13}$$

Generally, the parameter μ can be tuned via cross validation, though in our experience, the results from minimizing the loss function in Eq. (13) did not depend sensitively on the value of μ. In practice, the value $\mu = 0.5$ worked well.

The competing terms in Eq. (13) are analogous to those in the loss function for learning in SVMs (Schölkopf and Smola, 2002). In both loss functions, one term penalizes the norm of the "parameter" vector (i.e., the weight vector of the maximum margin hyperplane, or the linear transformation in the distance metric), while the other incurs the hinge loss. Just as the hinge loss in SVMs is only triggered by examples near the decision boundary, the hinge loss in Eq. (13) is only triggered by differently labeled examples that invade each other's neighborhoods. Both loss functions in SVMs and LMNN can be rewritten to depend on the input vectors only through their inner products. Working with the inner product matrix directly allows the application of the *kernel trick*; see section 5.3. Finally, as in SVMs, we can formulate the minimization of the loss function in Eq. (13) as a convex optimization. This last point will be developed further in section 3.4.

Our framework for distance metric learning provides an alternative to the earlier approach of NCA (Goldberger et al., 2005) described in section 2.4. We briefly compare the two approaches at a high level. Both LMNN and NCA are designed to learn a Mahalanobis distance metric over the input space that improves kNN classification at test time. Though test examples are not available during training, the learning algorithms for LMNN and NCA are based on training in "simulated" test conditions. Neither approach directly minimizes the leave-one-out error[1] for kNN classification over the training set. The leave-one-out error is a piecewise constant but non-smooth function of the linear transformation $\mathbf{L}$, making it difficult to minimize directly. NCA uses stochastic neighborhood assignment to construct a smooth loss function, thus circumventing this problem. LMNN uses the hinge loss to construct an upper bound on the leave-one-out error for kNN classification; this upper bound is continuous and similarly well behaved for standard gradient-based methods. In NCA, it is not necessary to select a fixed number k of target neighbors in advance of the optimization. Because the objective function for NCA is not convex, however, the initial conditions for the Mahalanobis metric implicitly favor the preservation of certain neighborhoods over others. By contrast, in LMNN, the target neighborhoods must be explicitly specified. A potential advantage of LMNN is that the required optimization can be formulated as an instance of semidefinite programming.

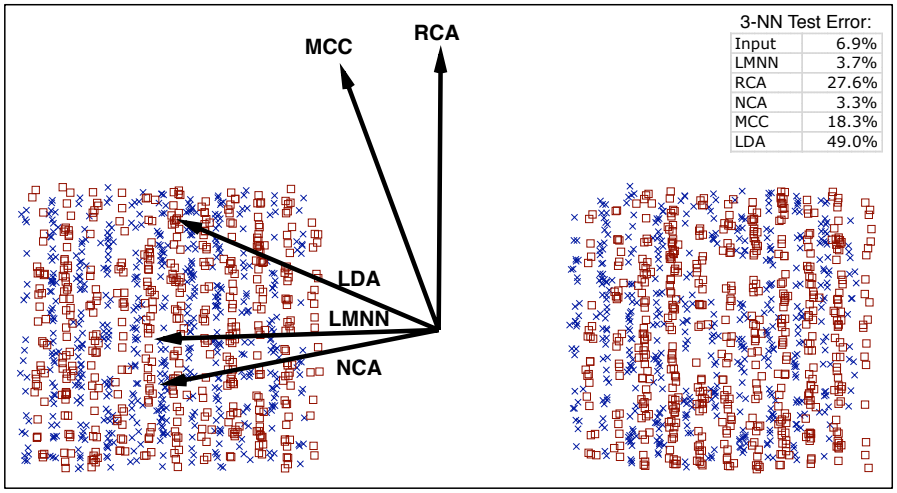

Figure 2: A toy data set for distance metric learning, with $n = 2000$ data points sampled from a bi-modal distribution. Within each mode, examples from two classes are distributed in alternating vertical stripes. The figure shows the dominant axis extracted by several different algorithms for distance metric learning. Only NCA and LMNN reduce the 1-NN classification error on this data set; the other algorithms actually increase the error by focusing on global versus local distances.

3.3 Local Versus Global Distances

We emphasize that the loss function for LMNN classification only penalizes large distances between target neighbors as opposed to all examples in the same class. The toy data set in Fig. 2 illustrates the potential advantages of this approach. The data was generated by sampling $n = 2000$ data points from two classes in a zebra striped pattern; additionally, the data for each class was generated in two sets of stripes displaced by a large horizontal offset. As a result, this data set has the property that within-class variance is much larger in the horizontal direction than the vertical direction; however, local class membership is much more reliably predicted by examples that are nearby in the vertical direction.

Algorithms such as LMNN and NCA perform very differently on this data set than algorithms such as LDA, RCA, and MCC. In particular, LMNN and NCA adapt to the local striped structure in the data set and learn distance metrics that significantly reduce the kNN error rate. By contrast, LDA, RCA, and MCC attempt to shrink distances between all examples in the same class and actually increase the kNN error rate as a result. Though this data set is especially contrived, it illustrates in general the problems posed by classes with multimodal support. Such classes violate a basic assumption behind metric learning algorithms that attempt to shrink global distances between all similarly labeled examples.

1. This is the number of training examples that *would have* been mislabeled by kNN classification if their label was in fact unknown.

3.4 Convex Optimization

The loss function in Eq. (13) is not convex in the matrix elements of the linear transformation $\mathbf{L}$. To minimize this loss function, one straightforward approach is gradient descent in the elements of $\mathbf{L}$. However, such an approach is prone to being trapped in local minima. The results of this form of gradient descent will depend in general on the initial estimates for $\mathbf{L}$. Thus they may not be reproducible across different problems and applications.

We can overcome these difficulties by reformulating the optimization of Eq. (13) as an instance of semidefinite programming (Boyd and Vandenberghe, 2004). A semidefinite program (SDP) is a linear program that incorporates an additional constraint on a symmetric matrix whose elements are linear in the unknown variables. This additional constraint requires the matrix to be positive semidefinite, or in other words, to only have nonnegative eigenvalues. This matrix constraint is nonlinear but convex, so that the overall optimization remains convex. There exist provably efficient algorithms to solve SDPs (with polynomial time convergence guarantees).

We begin by reformulating Eq. (13) as an optimization over positive semidefinite matrices. Specifically, as described in Eq. (2), we work in terms of the new variable $\mathbf{M} = \mathbf{L}^\top \mathbf{L}$. With this change of variable, we can rewrite the squared distances that appear in the loss function using Eq. (3). Recall that $\mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j)$ denotes the squared distance with respect to the Mahalanobis metric $\mathbf{M}$. As shown in section 2.1, this distance is equivalent to the Euclidean distance after the mapping $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$. Substituting Eq. (3) into Eq. (13), we obtain the loss function:

$$\varepsilon(\mathbf{M}) = (1-\mu) \sum_{i,\, j \rightsquigarrow i} \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) + \mu \sum_{i,\, j \rightsquigarrow i} \sum_{l} (1 - y_{il}) \left[1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_l)\right]_+ . \tag{14}$$

With this substitution, the loss function is now expressed over positive semidefinite matrices $\mathbf{M} \succeq 0$, as opposed to real-valued matrices $\mathbf{L}$. Note that the constraint $\mathbf{M} \succeq 0$ must be added to the optimization to ensure that we learn a well-defined pseudometric.

The loss function in Eq. (14) is a piecewise linear, convex function of the elements in the matrix $\mathbf{M}$. In particular, the first term in the loss function (penalizing large distances between target neighbors) is linear in the elements of $\mathbf{M}$, while the second term (penalizing impostors) is derived from the convex hinge loss. To formulate the optimization of Eq. (14) as an SDP, however, we need to convert it into a more standard form.

An SDP is obtained by introducing slack variables which mimic the effect of the hinge loss. In particular, we introduce nonnegative slack variables $\{\xi_{ijl}\}$ for all triplets of target neighbors ($j \rightsquigarrow i$) and impostors $\vec{x}_l$. The slack variable $\xi_{ijl} \geq 0$ is used to measure the amount by which the large margin inequality in Eq. (10) is violated. Using the slack variables to monitor these margin violations, we obtain the SDP:

Minimize $(1-\mu) \sum_{i,\, j \rightsquigarrow i} (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j) + \mu \sum_{i,\, j \rightsquigarrow i,\, l} (1 - y_{il}) \xi_{ijl}$ **subject to:**

(1) $(\vec{x}_i - \vec{x}_l)^\top \mathbf{M} (\vec{x}_i - \vec{x}_l) - (\vec{x}_i - \vec{x}_j)^\top \mathbf{M} (\vec{x}_i - \vec{x}_j) \;\geq\; 1 - \xi_{ijl}$

(2) $\xi_{ijl} \geq 0$

(3) $\mathbf{M} \succeq 0$**.**

While SDPs in this form can be solved by standard solver packages, general-purpose solvers tend to scale poorly in the number of constraints. For this work, we implemented our own special-purpose solver, exploiting the fact that most of the slack variables $\{\xi_{ijl}\}$ never attain positive values. The slack variables $\{\xi_{ijl}\}$ are sparse because most inputs $\vec{x}_i$ and $\vec{x}_l$ are well separated relative to the

distance between $\vec{x}_i$ and any of its target neighbors $\vec{x}_j$. Such triplets do not incur a positive hinge loss, resulting in very few *active* constraints in the SDP. Thus, a great speedup can be achieved by solving an SDP that only monitors a fraction of the margin constraints, then using the resulting solution as a starting point for the actual SDP of interest.

Our solver was based on a combination of sub-gradient descent in both the matrices $\mathbf{L}$ and $\mathbf{M}$, the latter used mainly to verify that we had reached the global minimum. We projected updates in $\mathbf{M}$ back onto the positive semidefinite cone after each step. Alternating projection algorithms provably converge (Vandenberghe and Boyd, 1996), and in this case our implementation[2] worked much faster than generic solvers. For a more detailed description of the solver please see appendix A.

3.5 Energy Based Classification

The matrix $\mathbf{M}$ that minimizes the loss function in Eq. (14) can be used as a Mahalanobis distance metric for kNN classification. However, it is also possible to use the loss function directly as a so-called "energy-based" classifier. This use is inspired by previous work on energy-based models (Chopra et al., 2005).

Energy-based classification of a test example is done by considering it as an extra training example and computing the loss function in Eq. (14) for every possible label y_t. In particular, for a test example $\vec{x}_t$ with hypothetical label y_t, we locate k (similarly labeled) target neighbors (as determined by Euclidean distance to $\vec{x}_t$ or other a priori considerations) and then compute both terms in Eq. (14) given the already estimated Mahalanobis metric $\mathbf{M}$. For the first term, we accumulate the squared distances to the k target neighbors of $\vec{x}_t$. For the second term, we accumulate the hinge loss over all impostors (i.e., differently labeled examples) that invade the perimeter around $\vec{x}_t$ as determined by its target neighbors; we also accumulate the hinge loss for differently labeled examples whose perimeters are invaded by $\vec{x}_t$. Finally, the test example is classified by the hypothetical label that minimizes the combination of these terms:

$$\begin{aligned} y_t \quad = \quad \text{argmin}_{y_t} \Bigg\{ & (1-\mu) \sum_{j \rightsquigarrow t} \mathcal{D}_{\mathbf{M}}(\vec{x}_t, \vec{x}_j) + \mu \sum_{j \rightsquigarrow t, l} (1-y_{tl}) \left[1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_t, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_t, \vec{x}_l)\right]_+ \\ & + \mu \sum_{i, j \rightsquigarrow i} (1-y_{it}) \left[1 + \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_j) - \mathcal{D}_{\mathbf{M}}(\vec{x}_i, \vec{x}_t)\right]_+ \Bigg\}. \end{aligned} \tag{15}$$

Note that the relation $j \rightsquigarrow t$ in this criterion depends on the value of y_t. As shown in Fig. 3, energy-based classification with this assignment rule generally leads to further improvements in test error rates. Often these improvements are significantly beyond those already achieved by adopting the Mahalanobis distance metric $\mathbf{M}$ for kNN classification.

4. Results

We evaluated LMNN classification on nine data sets of varying size and difficulty. Some of these data sets were derived from collections of images, speech, and text, yielding very high dimensional inputs. In these cases, we used PCA to reduce the dimensionality of the inputs before training LMNN classifiers. Pre-processing the inputs with PCA helped to reduce computation time and avoid overfitting. Table 1 compares the different data sets in detail.

2. A matlab implementation is currently available at `http://www.weinbergerweb.net/Downloads/LMNN.html`.

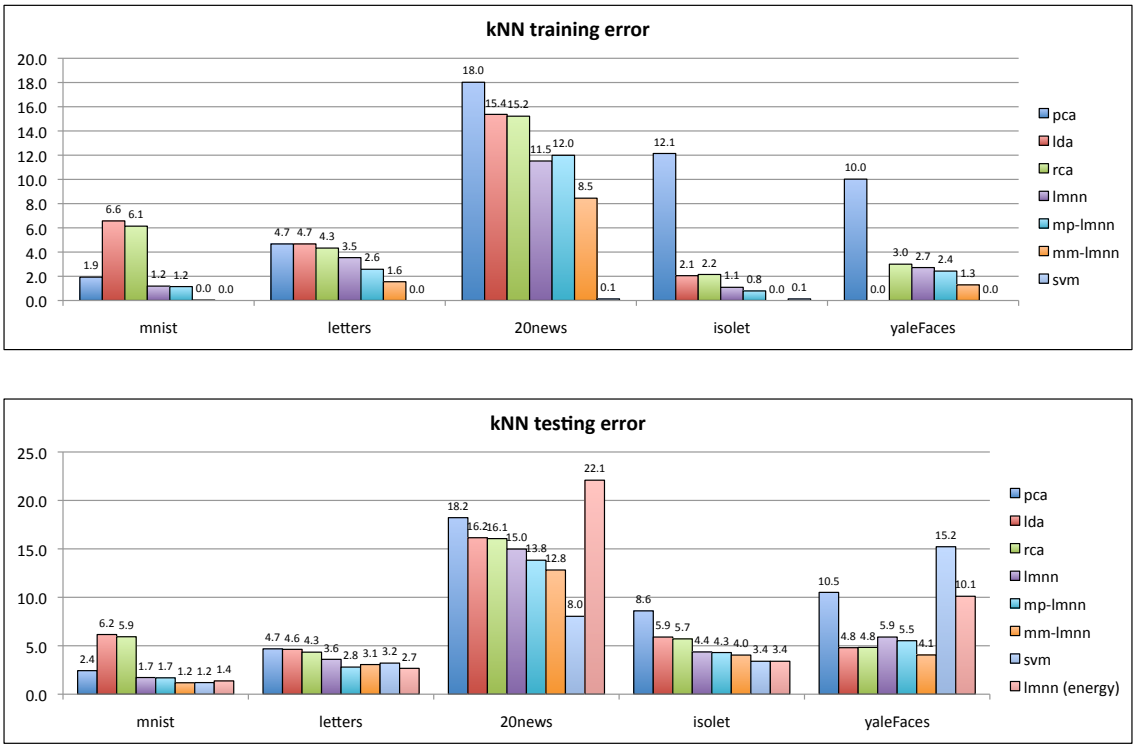

Figure 3: Training and test results on the five largest data sets, preprocessed in different ways, and using different variants of *k*NN classification. We compared principal component analysis (pca), linear discriminant analysis (lda), relevant component analysis (rca), large margin nearest neighbor classification (lmnn), lmnn with multiple passes (mp-lmnn), lmnn with multiple metrics (mm-lmnn), multi-class support vector machines (svm), lmnn classification with the energy based decision rule (lmnn (energy)). All variations of lmnn, rca and lda were applied after pre-processing with pca for general noise reduction. See text and Table 1 for details. The lmnn results consistently outperform pca and lda. The multiple metrics version of lmnn (mm-lmnn) is comparable with multiclass svm on most data sets (with 20news and yaleFaces as only exceptions).

Experimental results were obtained by averaging over multiple runs on randomly generated $70/30$ splits of each data set. This procedure was followed with two exceptions: no averaging was done for the Isolet and MNIST data sets, which have pre-defined training/test splits. For all experiments reported in this paper, the number of target neighbors k was set to $k=3$, and the weighting parameter μ in Eqs. (14-15) was set to $\mu=0.5$. Though we experimented with different settings, the results from LMNN classification appeared fairly insensitive to the values of these parameters.

The main results on the five largest data sets are shown in Fig. 3. (See Table 1 for a complete listing of results, including those for various extensions of LMNN classification described in section 5.) All training error rates reported are leave-one-out estimates. To break ties among different

classes from the kNN decision rule, we repeatedly reduced the neighborhood size, ultimately classifying (if necessary) by just the $k = 1$ nearest neighbor. We begin by reporting overall trends, then discuss the results on individual data sets in more detail.

The first general trend is that LMNN classification using Mahalanobis distances consistently improves on kNN classification using Euclidean distances. In general, the Mahalanobis metrics learned by semidefinite programming led to significant improvements in kNN classification, both in training and testing.

A second general trend is that the energy-based decision rule described in section 3.5 leads to further improvements over the (already improved) results from kNN classification using Mahalanobis distances. In particular, better performance was observed on most of the large data sets. The results are shown in Fig. 3.

A third general trend is that LMNN classification works better with PCA than LDA when some form of dimensionality reduction is required for preprocessing. Table 1 shows the results of LMNN classification on inputs whose dimensionality was reduced by LDA. While pre-processing by LDA helps on some data sets (e.g., wine, yale faces), it generally leads to worse results than pre-processing by PCA. On some data sets, moreover, it leads to drastically worse results (e.g., olivetti faces, MNIST). Consequently we used PCA as a pre-processing step for all subsequent experiments throughout this paper.

A fourth general trend is that LMNN classification yields larger improvements on larger data sets. Though we do not have a formal analysis that accounts for this observation, we can provide the following intuitive explanation. One crucial aspect of the optimization in LMNN classification is the choice of the *target neighbors*. In all of our experiments, we chose the target neighbors based on Euclidean distance in the input space (after dimensionality reduction by PCA or LDA). This choice was a simple heuristic used in the absence of prior knowledge. However, the quality of this choice presumably depends on the sample density of the data set. In particular, as the sample density increases, we suspect that more reliable discriminative signals can be learned from target neighbors chosen in this way. The experimental results bear this out.

Finally, we compare our results to those of competing methods. We take multi-class SVMs (Crammer and Singer, 2001) as providing a fair representation of the state-of-the-art. On each data set (except MNIST), we trained multi-class SVMs using linear, polynomial and RBF kernels and chose the best kernel with cross validation. On MNIST, we used a non-homogeneous polynomial kernel of degree four, which gave us our best results, as also reported in LeCun et al. (1995). The results of the energy-based LMNN classifier are very close to those of state-of-the-art multi-class SVMs: better on some data sets, worse on others. However, consistent improvement over multi-class SVMs was obtained by a multiple-metric variant of LMNN, discussed in section 5.2. This multi-metric extension outperformed SVMs on three of the five large data sets; see Fig. 3. On the only data set with a large performance difference, 20-newsgroups, the multi-class SVMs benefited from training in the original $d = 20000$ dimensional input space, whereas the LMNN classifiers were trained only on the input's leading $d = 200$ principal components. Based on these results, in section 7, we suggest some applications that seem particularly well suited to LMNN classification, though poorly suited to SVMs. These are applications with moderate input dimensionality, but large numbers of classes.

To compare with previous work, we also evaluated RCA (Shental et al., 2002), LDA (Fisher, 1936) and NCA (Goldberger et al., 2005) on the same data sets. For NCA and RCA, we used the code provided by the authors; however, the NCA code ran out of memory on the larger data sets.

Table 1 shows the results of all algorithms on small and larger data sets. LMNN outperforms these other methods for distance metric learning on the four largest data sets. In terms of running times, RCA is by far the fastest method (since its projections can be computed in closed form), while NCA is the slowest, mainly due to the $O(n^2)$ normalization of its softmax probability distributions. Although the optimization in LMNN naively scales as $O(n^2)$, in practice it can be accelerated by various efficiency measures: Appendix A discusses our semidefinite programming solver in detail. We did also include the results of MCC (Xing et al., 2002); however, the code provided by the authors could only handle a few of the small data sets. As shown in Table 1, on those data sets it resulted in classification rates generally higher than NCA.

The results of experiments on particular data sets provide additional insight into the performance of LMNN classification versus competing methods. We give a more detailed overview of these experiments in what follows.

4.1 Small Data Sets with Few Classes

The wine, iris, and bal data sets are small in size, with less than 500 training examples. Each of these data sets has three classes. The data sets are available from the UCI Machine Learning Repository.[3] On data sets of this size, a distance metric can be learned in a matter of seconds. The results in Table 1 were averaged over 100 experiments with different random $70/30$ splits of each data set.

On these data sets, LMNN classification improves on kNN classification with a Euclidean distance metric. These results could potentially be improved further with better measures against overfitting (such as regularization). Table 1 also compares the results from LMNN classification to other competing methods. Here, the results are somewhat variable; compared to NCA, RCA, LDA, and multiclass SVMs, LMNN fares better in some cases, worse in others. We mainly report these results to facilitate direct comparisons with previously published work. However, the small size of these data sets makes it difficult to assess the significance of these results. Moreover, these data sets do not represent the regime in which we expect LMNN classification to be most useful.

4.2 Face Recognition

The Olivetti face recognition data set[4] contains 400 grayscale images of 40 subjects in 10 different poses. We downsampled the images to 38×31 pixels and used PCA to further reduce the dimensionality, projecting the images into the subspace spanned by the first 200 eigenfaces (Turk and Pentland, 1991). Training and test sets were created by randomly sampling 7 images of each subject for training and 3 images for testing. The task involved 40-way classification—essentially, recognizing a face from an unseen pose. Table 1 shows the improvements due to LMNN classification. Fig. 4 illustrates the improvements more graphically by showing how the $k = 3$ nearest neighbors change as a result of learning a Mahalanobis metric. (Although the algorithm operated on downsampled, projected images, for clarity the figure shows the original images.)

The (extended) Yale face data set contains $n = 2414$ frontal images of 38 subjects. For each subject, there are 64 images taken under extreme illumination conditions. (A few subjects are represented with fewer images.) As for the Olivetti data set, we preprocessed the images by downsampling and projecting them onto their leading 200 principal components. To reduce the impact of the very high variance in illumination, we followed the standard practice of discarding the leading 5

3. Available at `http://www.ics.uci.edu/$\sim$mlearn/MLRepository.html`.
4. Available at `http://www.uk.research.att.com/facedatabase.html`.

eigenvectors. Results from LMNN classification were averaged over 10 runs of $70/30$ splits. Each split was obtained by randomly selecting 45 images of each subject for training and 19 images for testing. This protocol ensured that the training examples were evenly distributed across the relatively large number of classes. To guard against overfitting, we employed a validation set consisting of 30% of the training data and stopped the training early when the lowest classification error on the validation set was reached. On this data set, Fig. 3 shows that the LMNN metric outperforms the Euclidean metric and even improves on multiclass SVMs. (Particularly effective on this data set, though, is the simple strategy of LDA.)

Figure 4: Test images from the Olivetti face recognition data set (*top row*). The middle row shows images from the same class that were among the 3-NN under the learned Mahalanobis metric (after training) but not among the original 3-NN under the Euclidean metric (before training). The bottom row shows impostors under the Euclidean metric that were no longer inside the local neighborhoods under the Mahalanobis metric.

4.3 Spoken Letter Recognition

The Isolet data set from the UCI Machine Learning Repository contains 6238 examples and 26 classes corresponding to letters of the alphabet. We reduced the input dimensionality (originally at 617) by projecting the data onto its leading 172 principal components—enough to account for 95% of its total variance. On this data set, Dietterich and Bakiri report test error rates of 4.2% using nonlinear backpropagation networks with 26 output units (one per class) and 3.3% using nonlinear backpropagation networks with a 30-bit error correcting code (Dietterich and Bakiri, 1995). LMNN with energy-based classification obtains a test error rate of 3.4%.

4.4 Letter Recognition

The letter recognition data set was also taken from the UCI Machine Learning Repository. It contains randomly distorted images of the 26 letters in the English alphabet in 20 different fonts. The features consist of 16 attributes, such as height, width, correlations of axes and others.[5] It is inter-

5. Full details on the data set can be found at `http://www.ics.uci.edu/$\sim$mlearn/databases/letter-recognition/letter-recognition.names`.

esting that LMNN with energy-based classification significantly outperforms other variants of *k*NN classification on this data set.

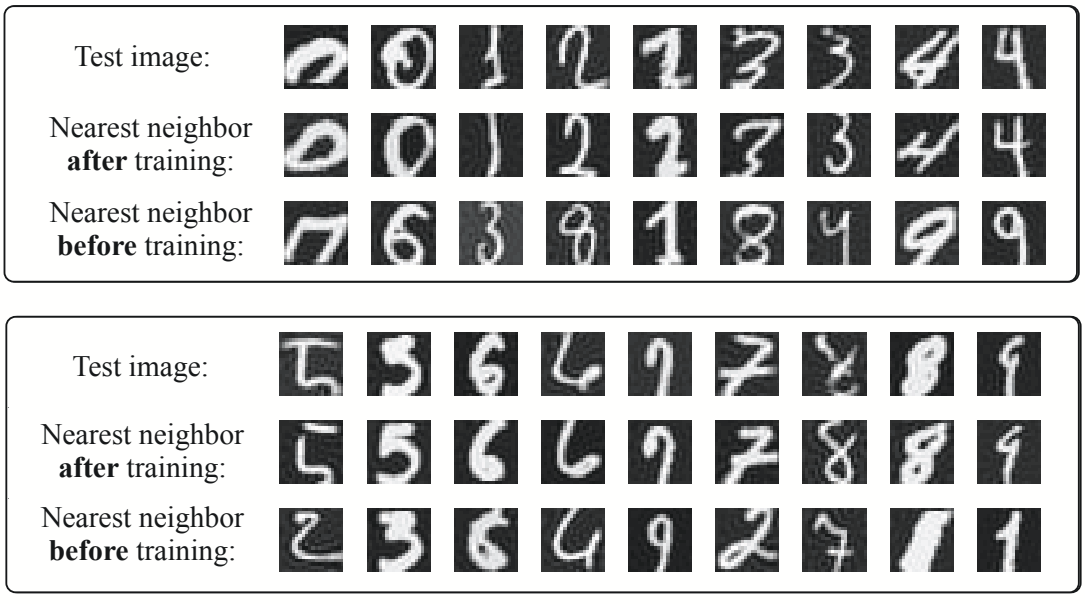

Figure 5: Images from the MNIST data set, along with nearest neighbors before and after training.

4.5 Text Categorization

The 20-newsgroups data set consists of posted articles from 20 newsgroups, with roughly 1000 articles per newsgroup. We used the 18828-version of the data set[6] in which cross-postings are removed and some headers stripped out. The data set was tokenized using the rainbow package (McCallum, 1996). Each article was initially represented by a word-count vector for the 20,000 most common words in the vocabulary. These word-count vectors were then reduced in dimensionality by projecting them onto their leading 200 principal components. The results in Fig. 3 were obtained by averaging over 10 runs with 70/30 splits for training and test data. The best result for LMMN on this data set improved significantly over *k*NN classification using Euclidean distances and PCA (with 14.98% versus 48.57% and 18.22% test error rates). LMNN was outperformed by multiclass SVM (Crammer and Singer, 2001), which obtained a 8.0% test error rate using a linear kernel and 20000 dimensional inputs.[7]

4.6 Handwritten Digit Recognition

The MNIST data set of handwritten digits[8] has been extensively benchmarked (LeCun et al., 1995). We deskewed the original 28×28 grayscale images, then reduced their dimensionality by projecting them onto their leading 164 principal components (enough to capture 95% of the data's overall

6. Available at `http://people.csail.mit.edu/jrennie/20Newsgroups/`.
7. Results vary from previous work (Weinberger et al., 2006) due to different pre-processing.
8. Available at `http://yann.lecun.com/exdb/mnist/`.

Benchmark test error rates

statistics	**mnist**	**letters**	**20news**	**isolet**	**yFaces**	**bal**	**oFaces**	**wine**	**iris**
# inputs	70000	20000	18827	7797	2414	535	400	152	128
# features	784	16	20000	617	8064	4	200	13	4
# reduced dimensions	164	16	200	172	300	4	200	13	4
# training examples	60000	14000	13179	6238	1690	375	280	106	90
# testing examples	10000	6000	5648	1559	724	161	120	46	38
# classes	10	26	20	26	38	3	40	3	3
# of train/test splits	1	10	10	1	10	100	100	100	100
% validation	0	0	0	0	30	0	30	0	30
kNN									
Euclidean	2.12	4.68	48.57	8.98	29.19	18.33	6.03	25.00	4.87
PCA	2.43	4.68	18.22	8.60	10.79	18.33	2.80	25.00	4.87
LDA	6.16	4.63	16.15	5.90	4.80	10.82	10.01	2.17	4.00
RCA	5.93	4.34	16.06	5.71	4.83	12.31	10.02	2.28	3.71
MCC	N/A	N/A	N/A	N/A	N/A	15.66	15.91	30.96	3.55
NCA	N/A	N/A	N/A	N/A	N/A	5.33	2.60	28.67	4.32
LMNN									
PCA	1.72	3.60	14.98	4.36	5.90	11.16	3.28	8.72	4.37
LDA	6.16	3.61	16.98	5.84	5.08	10.84	40.72	**2.11**	3.79
LMNN (energy)	1.37	**2.67**	22.09	**3.40**	10.11	9.14	3.16	7.67	3.68
LMNN (multiple passes)	1.69	2.80	13.83	4.30	5.52	5.86	4.83	7.59	4.26
LMNN (multiple metrics)	**1.18**	3.06	12.82	4.04	**4.05**	10.72	3.11	8.72	4.66
solver statistics									
CPU time (1M)	3h 25m	2m	70m	20m	8m	6s	66s	14s	2s
CPU time (MM)	8h 43m	14m	74m	84m	14m	8s	149s	16s	5s
# active constraints (1M)	540037	135715	676482	64396	86994	41522	3843	10194	574
# active constraints (MM)	305114	18588	101803	135832	30135	31717	70	748	1548
multiclass SVM	1.20	3.21	**8.04**	**3.40**	15.22	**1.92**	**1.90**	22.24	**3.45**

larger data sets **smaller data sets**

Table 1: Results and statistics from all experiments. The data sets are sorted by largest to smallest from left to right. The table shows data statistics and error rates from different variants of LMNN training (single-pass, multi-pass, multi-metric), testing (*k*NN decision rule, energy-based classification), and preprocessing (PCA, LDA). Results from RCA, NCA and multiclass support vector machines (SVMs) are also provided for comparison. See section 5 for discussion of multi-pass and multi-metric LMNN training.

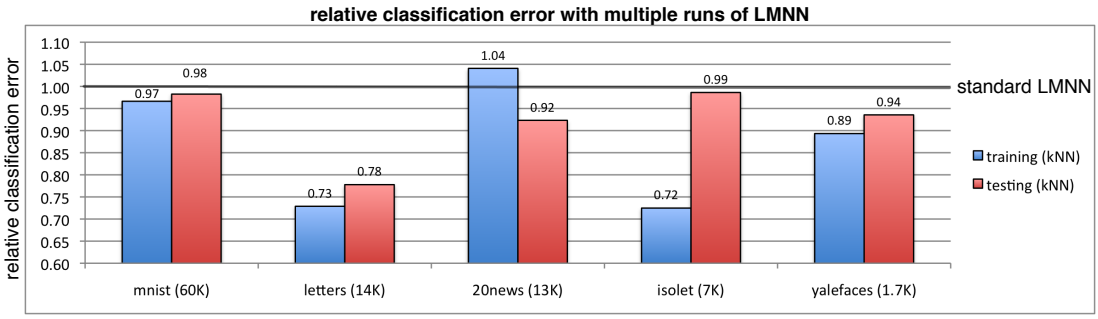

Figure 6: The relative change of the 3-NN classification error after multiple runs of LMNN over a single run of LMNN.

variance). Energy-based LMNN classification yielded a test error rate at 1.4%, cutting the baseline kNN error rate by over one-third. Other comparable benchmarks (LeCun et al., 1995) (not exploiting additional prior knowledge) include multilayer neural nets at 1.6% and SVMs at 1.2%. Fig. 5 shows some digits whose nearest neighbor changed as a result of learning, from a mismatch using Euclidean distances to a match using Mahalanobis distances. Table 1 reveals that the LMNN error can be further reduced by learning a different distance metric for each digit class. This is discussed further in section 5.2.

5. Extensions

In this section, we investigate four extensions designed to improve LMNN classification. Section 5.1 examines the impact of multiple consecutive applications of LMNN on one data set. Section 5.2 shows how to learn multiple (locally linear) metrics instead of a single global metric. Section 5.3 discusses how to "kernelize" the algorithm for LMNN classification and reviews complementary work by Torresani and Lee (2007). Finally, section 5.4 investigates the use of LMNN as a method for supervised dimensionality reduction.

5.1 Multi-pass LMNN

One potential weakness of LMNN is that target neighbors must be a priori specified. In the absence of prior knowledge, a default choice is to use Euclidean distances to determine target nearest neighbors. While the target nearest neighbors are fixed during learning, however, the actual nearest neighbors may change as a result of the linear transformation of the input space. These changes suggest an iterative approach, in which the Mahalanobis distances learned in one application (or "pass") of LMNN are used to determine the target nearest neighbors in a subsequent run of the algorithm. More formally, let $\mathbf{L}_p$ be the transformation matrix obtained from the p^{th} pass of LMNN. For the $(p+1)^{th}$ pass, we can assign target neighbors using the Euclidean distance metric after the linear transformation $\vec{x}_i \rightarrow \mathbf{L}_p\mathbf{L}_{p-1}...\mathbf{L}_1\mathbf{L}_0\vec{x}_i$ (with $\mathbf{L}_0 = \mathbf{I}$).

To evaluate this approach, we performed multiple passes of LMNN on all the data sets from Table 1. The parameter k was set to $k = 3$. Figure 6 shows the relative improvements in kNN classi-

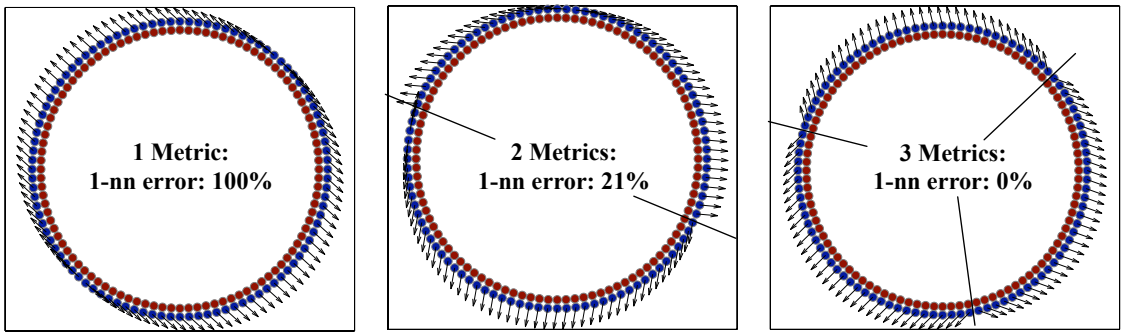

Figure 7: A synthetic data set to illustrate the potential of multiple metrics. The data set consists of inputs sampled from two concentric circles, each of which defines a different class membership. LMNN training was used to estimate one global metric, as well as multiple local metrics. *Left:* a single linear metric cannot model the non-linear decision boundary. The leave-one-out (LOO) error is 100%. *Middle:* if the data set is divided into two clusters (by k-means), and a local metric learned within each cluster, the error rate drops drastically. *Right:* the use of three metrics reduces the LOO-error on the training set to zero. The principal directions of individual distance metrics are indicated by arrows.

fication error rates on the five largest data sets. (Here, a value of one indicates that multiple passes of LMNN did not change the error rate, while a value less than one indicates an improvement.) On these data sets, multiple passes of LMMN were generally helpful, sometimes significantly improving the results. On smaller data sets, though, the multi-pass strategy seemed prone to overfit. Table 1 shows the absolute results on all data sets from multiple passes of LMNN (indicated by MP-LMNN).

A better strategy for choosing target neighbors remains an open question. This aspect of LMNN classification differs significantly from NCA, which does not require the choice of target neighbors. In fact, NCA also determines the effective neighborhood size as part of its optimization. On the other hand, the optimization in NCA is not convex; as such, the initial conditions implicitly specify a basin of attraction that determines the final result. In LMNN classification, the target neighbors are fixed in order to obtain a convex optimization. This trade-off is reminiscent of other convex relaxations of computationally hard problems in machine learning.

5.2 Multi-metric LMNN

On some data sets, a global linear transformation of the input space may not be sufficiently powerful to improve *k*NN classification. Figure 7 shows an example of a synthetic data set for which a single metric is not sufficient. The data set consists of inputs sampled from two concentric circles, each of which defines a different class membership. Global linear transformations cannot improve the accuracy of kNN classification of this data set. In general, highly nonlinear multiclass decision boundaries may not be well modeled by a single Mahalanobis distance metric.

In these situations, one useful extension of LMNN is to learn multiple locally linear transformations instead of a single global linear transformation. In this section, we show how to learn different Mahalanobis distance metrics for different examples in the input space. The idea of learning locally linear distance metrics for kNN classification is at least a decade old (Hastie and Tibshirani, 1996). It has also been explored more recently in the context of metric learning for semi-supervised clustering (Bilenko et al., 2004). The novelty of our approach lies in learning these metrics specifically to maximize the margin of correct kNN classification. As a first step, we partition the training data into disjoint clusters using k-means, spectral clustering (Shi and Malik, 2000), or label information. (In our experience, the latter seems to work best.) 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 integrated training of local distance metrics also distinguishes our approach from earlier work (Hastie and Tibshirani, 1996).

Before developing this idea more formally, we first illustrate its potential in a toy setting—namely, on the data set in Fig. 7. For this data set, LMNN training was used to estimate one global metric, as well as multiple local metrics (as described below). Cluster boundaries in the input space were determined by the k-means algorithm. We measured the leave-one-out (LOO) training error (with $k = 1$ nearest neighbors) after learning one, two and three metrics. With one metric, the error was 100%; with two metrics, it dropped to 21%; finally, with three metrics, it vanished altogether. The figure illustrates how the multiple metrics adapt to the local structure of the class decision boundaries.

In order to learn different Mahalanobis metrics in different parts of the input space, we minimize a variation of the objective function in Eq. (14). We denote the different metrics by $\mathbf{M}^1, \ldots, \mathbf{M}^c$, where c is the number of clusters. If we partition the training examples by their class labels, then c also coincides with the number of classes; this was done for the remaining experiments in this section. In this case, as the cluster that contains $\vec{x}_i$ is indexed by its label y_i, we can refer to its metric as $\mathbf{M}^{y_i}$. We further define the cluster-dependent distance between two vectors $\vec{x}_i$ and $\vec{x}_j$ as:

$$\hat{\mathcal{D}}(\vec{x}_i, \vec{x}_j) = (\vec{x}_i - \vec{x}_j)^\top \mathbf{M}^{y_j} (\vec{x}_i - \vec{x}_j). \tag{16}$$

Note that this cluster-dependent measure of distance $\hat{\mathcal{D}}(\vec{x}_i, \vec{x}_j)$ is not symmetric with respect to its input arguments. In a slight abuse of terminology, however, we will continue to refer to Eq. (16) as a distance metric; the symmetry is not required for its use in kNN classification. To learn these metrics from data, we solve a modified version of the original SDP:

Minimize $(1-\mu)\sum_{i,j\rightsquigarrow i}(\vec{x}_i - \vec{x}_j)^\top \mathbf{M}^{y_j}(\vec{x}_i - \vec{x}_j) + \mu\sum_{j\rightsquigarrow i,l}(1-y_{il})\xi_{ijl}$
subject to:

(1) $(\vec{x}_i - \vec{x}_l)^\top \mathbf{M}^{y_l}(\vec{x}_i - \vec{x}_l) - (\vec{x}_i - \vec{x}_j)^\top \mathbf{M}^{y_j}(\vec{x}_i - \vec{x}_j) \;\geq\; 1 - \xi_{ijl}$

(2) $\xi_{ijl} \geq 0$

(3) $\mathbf{M}^i \succeq 0$ **for** $i = 1, \ldots, c$**.**

Note that all the matrices $\mathbf{M}^i$ are learned simultaneously by solving a single SDP. This approach ensures the consistency of distance computations in different clusters: for example, the distance from a test example to training examples with different labels. The integrated learning of different metrics is necessary to calibrate these distances on the same scale; if the metrics were

learned independently, then the distances computed by different metrics could not be meaningfully compared—obviously, a crucial requirement for *k*NN classification.

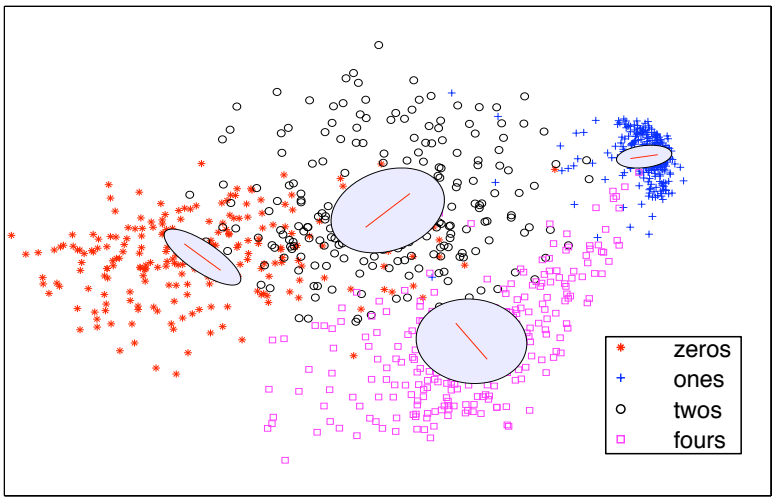

Figure 8: Multiple local distance metrics learned for a data set consisting of handwritten digits *four*, *two*, *one* and *zero*.

Fig. 8 illustrates the multiple metrics learned from an image data set of four different handwritten digits: *zero*, *one*, *four*, and *two*. The plot shows the first two principal components of the data. Only these principal components were used in training in order to yield an easily visualized solution. The solution can be visualized by illustrating the metrics as ellipsoids centered at the class means. The ellipsoids show the effect on a unit circle of each local linear transformation learned by LMNN. The line inside each ellipsoid indicates its principal axis.

We experimented with this multi-metric version of LMNN on all of the data sets from section 4. To avoid overfitting, we held out 30% of each data set's training examples and used them as a validation set. We learned one metric per class. To speed up training, we initialized the multi-metric optimization by setting each class-dependent metric to the solution from LMNN classification with a single global distance metric. Table 1 reports the error rates and other results from all these experiments (under "MM-LMNN"). The training times for MM-LMNN include the time required to compute the initial metric settings from the optimization in Eq. (14).

Fig. 9 shows the relative improvement in 3-NN classification error rates from multi-metric LMNN over standard LMNN on the five largest data sets. The multiple metrics variant improves over standard LMNN on every data set. The best result occurs on the MNIST handwritten digits data set, where MM-LMNN obtained a 1.18% *k*NN classification error rate, slightly outperforming multi-class SVMs. However, the improvement from multi-metric LMNN is not as consistently observed when the energy-based decision rule is used for classification.

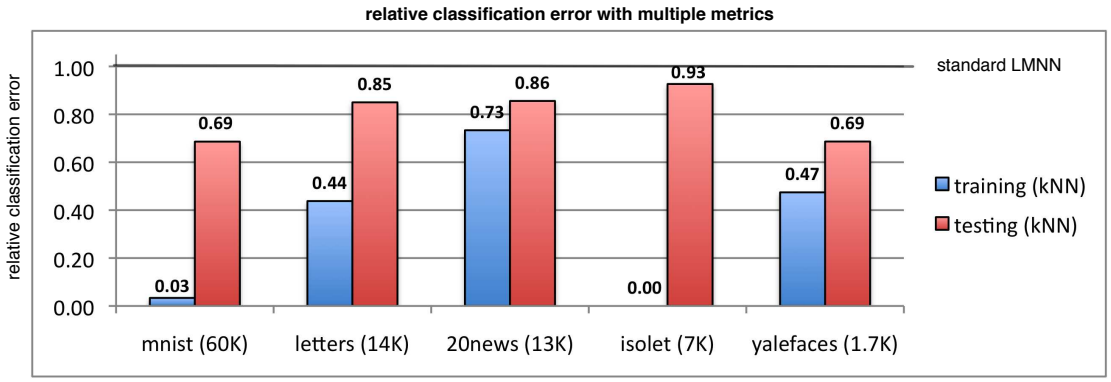

Figure 9: Relative improvement in $k-$NN classification error rates using multiple metrics over error rates using a single metric.

5.3 Kernel Version

LMNN can also be extended by using kernel methods (Schölkopf and Smola, 2002) to work in a nonlinear feature space, as opposed to the original input space. The idea of learning a kernel matrix has been explored in other contexts (Kwok and Tsang, 2003; Lanckriet et al., 2004; Varma and Ray, 2007), particularly large margin classification by support vector machines. This idea for LMNN has been investigated in detail by Torresani and Lee (2007). The "kernel trick" is used to map the inputs $\vec{x}_i$ into higher (possibly infinite) dimensional feature vectors $\Phi(\vec{x}_i)$. To avoid the computational cost of working directly with these feature vectors, they are only accessed through their inner products, which are pre-computed and stored in the kernel matrix:

$$\mathbf{K}_{ij} = \Phi(\vec{x}_i)^\top \Phi(\vec{x}_j).$$

Note how in Eq. (14), the inputs $\vec{x}_i$ are only accessed in terms of the distances in Eq. (3). Torresani and Lee (2007) considered Mahalanobis metrics of the form $\mathbf{M} = \sum_{lm} \mathbf{A}_{lm} \Phi(\vec{x}_l) \Phi(\vec{x}_m)^\top$, where the matrix $\mathbf{A}$ is constrained to be positive semidefinite. They showed that the gradient of Eq. (14) with respect to the matrix $\mathbf{A}$ can be written entirely in terms of the elements of the kernel matrix. Thus, a "kernelized" version of LMNN can be implemented efficiently in the same way as kernel PCA (Schölkopf et al., 1998), without ever working directly in the high dimensional feature space.

Torresani and Lee (2007) show that the kernelized version of LMNN can lead to significant further improvements, but at the cost of increased computation. The increased computation is due to the size of the matrix that must be learned in this setting: the matrix $\mathbf{A}$ has $O(n^2)$ elements instead of $O(d^2)$. (However, the kernel version could require less computation in applications where $n < d$.) More details on the kernelized version of LMNN can be found in their paper.

5.4 Dimensionality Reduction

Often it is useful to generate low dimensional representations of high dimensional data. These representations can be used to visualize the data and/or to accelerate algorithms whose time complexity

scales with the input dimensionality. In section 6, for example, we will investigate how to accelerate the *k*NN search in LMNN classification by mapping the training data into a low dimensional subspace.

Low dimensional representations of inputs can be derived from the linear transformation $\vec{x}_i \rightarrow \mathbf{L}\vec{x}_i$ in LMNN classification. This can be done in two ways. The first way is to project the *transformed* inputs onto their leading principal components. Note that if the inputs are whitened prior to optimizing Eq. (13), then these principal components are given simply by the leading eigenvectors of the square matrix $\mathbf{L}$. Another way to derive low dimensional representations is to build this goal explicitly into the optimization for LMNN classification. In particular, we can attempt to minimize Eq. (13) with respect to $\mathbf{L}$ (rather than with respect to $\mathbf{M} = \mathbf{L}^\top\mathbf{L}$) and constrain $\mathbf{L}$ to be rectangular of size $r \times d$, where r is the desired output dimensionality (presumed to be much smaller than the input dimensionality, d). The optimization in terms of $\mathbf{L}$ is not convex, but in practice (Torresani and Lee, 2007), it does not appear to suffer from very poor local minima. In the following section, we use and compare both these methods to build efficient tree data structures for LMNN classification.

6. Metric Trees

One inherent disadvantage of *k*NN search is its relatively high computational complexity at test time. The simplest brute-force way to locate a test example's nearest neighbors is to compute its distance to all the training examples. Such a naïve implementation has a test time-complexity of $O(nd)$, where n is the number of training examples, and d is the input dimensionality.

One way to accelerate *k*NN search is to rotate the input space such that the coordinate axes are aligned with the data's principal components. Such a rotation sorts the input coordinates by decreasing variance; see Section 2.2.1. This ordering can be used to prune unnecessary computations in *k*NN search. In particular, for any test example, a nearest neighbor query consists of computing the distance to each training example and comparing this distance to the k closest examples already located. The distance computation to a particular training example can be aborted upon determining that it lies further away than the k closest examples already located. When the coordinate axes are aligned with the principal components, this determination can often be made after examining just a few of the leading, load-bearing dimensions. We have used this optimization in our baseline implementation of *k*NN search.

Generally there are two major approaches to gain additional speed-ups. The first approach is to reduce the input dimensionality d. The Johnson-Lindenstrauss Lemma (Dasgupta and Gupta, 1999) states that n points can be mapped into a space of dimensionality $O(\frac{\log(n)}{\varepsilon^2})$ such that the distances between any two points changes only by a factor of $(1 \pm \varepsilon)$. Thus we can often reduce the dimensionality of the input data without distorting the nearest neighbor relations. (Note also that for *k*NN classification, we may tolerate inexact nearest neighbor computations if they do not lead to significant errors in classification.) The second approach to speed up *k*NN search is to build a sophisticated tree-based data structure for storing training examples. Such a data structure can reduce the nearest neighbor test time complexity in practice to $O(d \log n)$ (Beygelzimer et al., 2006). This latter method works best for low dimensional data. Fig. 10 compares a baseline implementation of *k*NN search versus one based on ball trees (Liu et al., 2005; Omohundro, 1987). Note how the speed-up from the ball trees is magnified by dimensionality reduction of the inputs.

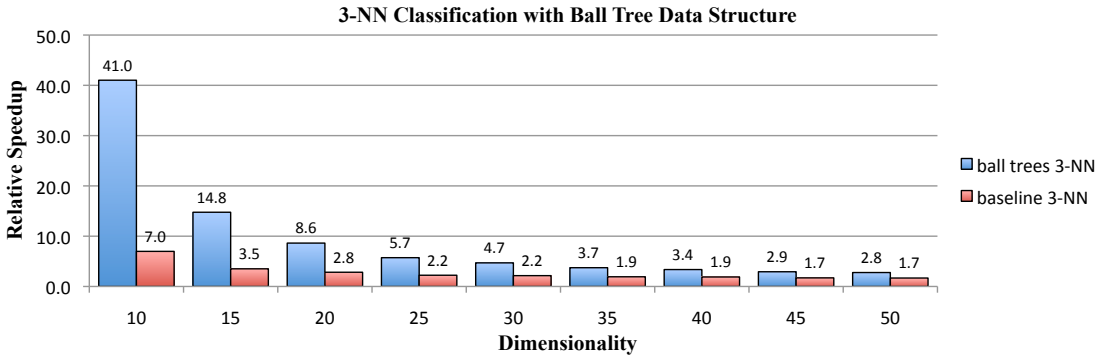

Figure 10: Relative speed-up for 3NN classification obtained from different orthogonal projections of MNIST handwritten digits onto their leading principal components. For these experiments, the $d = 784$ dimensional inputs from the raw images were projected onto the number of principal components shown on the x-axis. The figure compares the speed-ups when ball trees are used (blue) versus when ball trees are not used (red) in the lower dimensional space. Note how the gains from ball trees diminish with increasing dimensionality. All the NN computations in these experiments were accelerated by aligning the coordinate axes along principal components, as described in section 6.

In this section, we explore the use of ball trees for LMNN classification and dimensionality reduction. We find that ball trees can be used for both faster training and testing of LMNN classifiers.

6.1 Review of Ball Trees

Several authors have proposed tree-based data structures to speed up kNN search. Examples are kd-trees (Friedman et al., 1977), ball trees (Liu et al., 2005; Omohundro, 1987) and cover-trees (Beygelzimer et al., 2006). All these data structures exploit the same idea: to partition the input space data into hierarchically nested bounding regions. The bounding regions are set up to guarantee that the distance from a test example to a training example inside the bounding region is at least as large as the distance from the test example's to the region's boundary. Thus, for each test example, the training examples inside the bounding region can be ruled out as k nearest neighbors if k training examples have already been found that are closer than the region's boundary. In this case, the kNN search can proceed without explicitly computing the distances to training examples in the bounding region. This "pruning" of distance computations often leads to a significant speedup in kNN computation time.

We experimented with ball trees (Liu et al., 2005), in which the bounding regions are hyperspheres. Fig. 11 illustrates the basic idea behind ball trees. If a set S of training examples is encapsulated inside a ball with center $\vec{c}$ and radius r, such that $\forall \vec{x} \in S : \ \|\vec{x} - \vec{c}\| \leq r$, then for any test example $\vec{x}_t$ we can bound the distance to any training example inside the ball by the following expression:

$$\forall \vec{x}_i \in S \quad \|\vec{x}_t - \vec{x}_i\| \geq \max(\|\vec{x}_t - \vec{c}\|_2 - r, 0). \tag{17}$$

Ball trees exploit this inequality to build a hierarchical data structure. The data structure is based

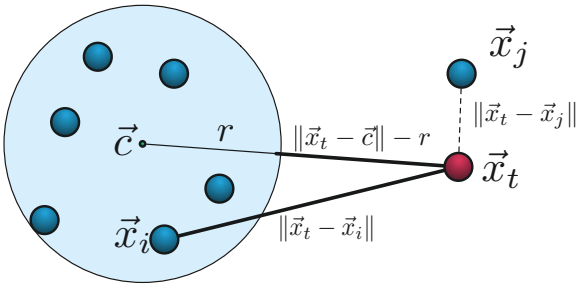

Figure 11: The basic idea behind ball trees: for any training example $\vec{x}_i$ inside the ball we can bound the distance $\|\vec{x}_t - \vec{x}_i\|_2$ from below using (17). If another training example $\vec{x}_j$ outsider the ball is already known to be closer than this bound to the test example $\vec{x}_t$, then the training examples inside the ball can be ruled out as nearest neighbors.

on recursively splitting the training examples into two disjoint sets. The sets are encapsulated by hyperspheres (or "balls") which may be partially overlapping. The training examples are recursively divided into smaller and smaller sets until no leaf set contains more than some predefined number of examples.

From this hierarchical data structure, the *k*-nearest neighbors of a test example can be found by a standard depth-first tree-based search. Recall that each node in the tree has an associated hypersphere that encloses the training examples stored by its descendants. The *k*NN search proceeds by traversing the tree and computing a test example's distance to the center of each node's hypersphere. The tree is traversed by greedily descending sub-trees in order of this distance. Before descending a subtree, however, Eq. (17) is checked to determine whether training examples in the subtree lie further away than the currently estimated *k*-nearest neighbors. If this is true, the sub-tree is pruned from the search without further computation. When a leaf node is reached, all the training examples at the leaf node are compared to the currently estimated *k*-nearest neighbors, and the estimates are updated as necessary. Note that ball trees support exact queries for *k*NN search.

As pointed out earlier, and as illustrated by Fig. 10, ball trees yield the largest gains in *k*NN search time for low dimensional data. When the data is high dimensional, the search is plagued by the so-called "curse of dimensionality" (Indyk and Motwani, 1998). In particular, the distances between high dimensional points tend to be more uniform, thereby reducing the opportunities for pruning subtrees in the depth-first search.

6.2 Ball Trees for LMNN Training

The most computationally intensive part of LMNN training is computing the gradient of the penalty for margin violations in Eq. (12). The gradient computation requires a search over all pairs of differently labeled examples to determine if any of them are "impostors" (see section 3.1) that incur margin violations. The solver described in appendix A reduces the number of these searches by maintaining an active list of previous margin violations. Nevertheless, this search scales $O(n^2 d)$, which is very computationally intensive for large data sets.

Ball trees can be used to further speed up the search for impostors. Recall how impostors were defined in section 3.1. For any training example $\vec{x}_i$, and for any similarly labeled example $\vec{x}_j$ that

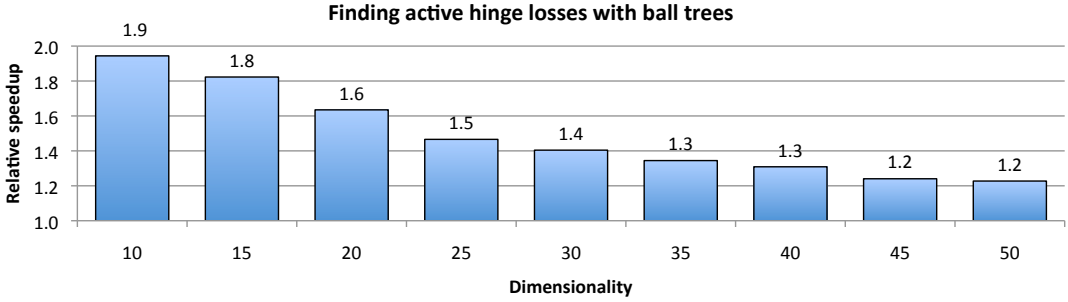

Figure 12: The relative speed-up obtained using ball trees to search for margin violations. The speed-up was measured on the MNIST data set of handwritten digits, with inputs of varying dimensionality derived from PCA. Note how the gains from ball trees diminish with increasing input dimensionality.

is one of its target k-nearest neighbors (with $j \rightsquigarrow i$), the impostors consist of all differently labeled examples $\vec{x}_l$ (with $y_{il} = 0$) that satisfy Eq. (10). Ball trees can be used to search for all training examples that meet this criterion. As in their use for kNN search, many subtrees in the depth-first search for impostors can be pruned: if for some ball the lower bound distances between examples is already greater than the right hand side of Eq. (10), then all the examples stored in the subtree can be ruled out as impostors. Note that for each training example $\vec{x}_i$, we only need to search for impostors among other training examples $\vec{x}_l$ that have a different class label (with $y_{il} = 0$). Thus, we build one ball tree data structure per class and perform a separate search for impostors in each class.

Fig. 12 shows the relative speed-up when ball trees are used to search for margin violations in LMNN classification. The figure shows results from experiments with the MNIST images of handwritten digits. For these experiments, the images were projected into subspaces of varying dimensionality using PCA. The gains from ball trees in this context are significant, though not as dramatic as those in Fig. 10 for simple kNN search. The lesser gains for LMNN classification can be attributed to the minimum enforced margin of unit distance, which sometimes causes a high number of sub-trees to be traversed. This effect is controlled by the relative magnitude of the unit margin; it can be partially offset by scaling the input data by a constant factor before training.

6.3 Ball Trees for LMNN Testing

Ball trees can also be used to accelerate kNN search at test time. We have observed earlier, though, that the speed-up from ball trees diminishes quickly as the input dimensionality increases; see Fig. 10. If very fast kNN classification using ball trees is desired on a large data set, then often it is necessary to work with a lower dimensional representation of the training examples.

The most commonly used methods for dimensionality reduction in ball trees are random projections and PCA. Neither of these methods, however, is especially geared to preserve the accuracy of kNN classification. There is an inherent trade-off between dimensionality reduction and nearest

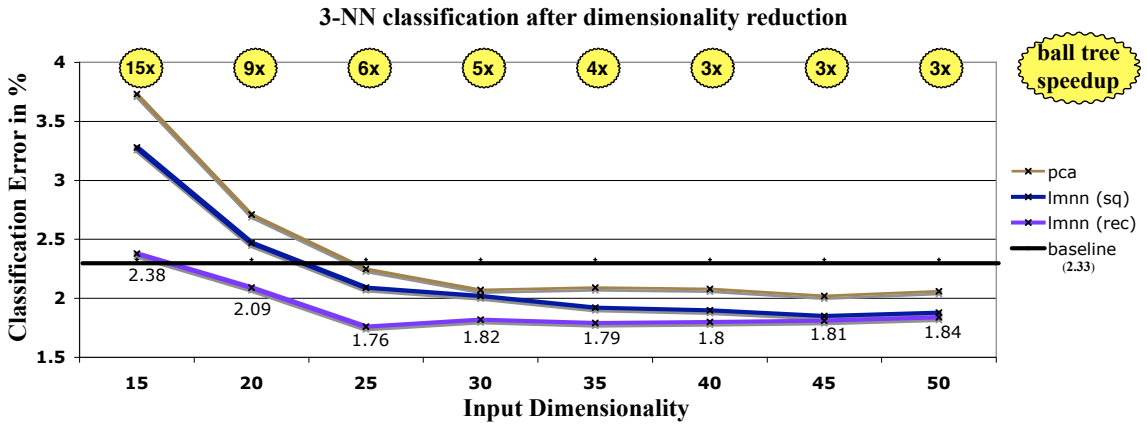

Figure 13: Graph of kNN classification error (with $k = 3$) on different low dimensional representations of the MNIST data set; see text for details. The speed-up from ball-trees is shown at the top of the graph.

neighbor preservation. Nearest neighbor relationships can change when the training examples are projected into a lower dimensional space, resulting in significantly worse kNN classification.

In this section, we explore how the distance metric learned for LMNN classification can be used for more effective dimensionality reduction in ball trees. In section 5.4, we described two different ways to derive low dimensional representations for LMNN classification. The first computed a low-rank approximation to the (generally full rank) matrix $\mathbf{L}$; the second directly learned a low-rank rectangular matrix $\mathbf{L}$ by optimizing the non-convex loss function in Eq. (13). For shorthand, we refer to these approaches for dimensionality reduction as LMNN-S and LMNN-R, denoting whether a square (S) or rectangular (R) matrix is learned to minimize the LMNN cost function. Fig. 13 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 $28\times$ 28) were projected onto their 350 leading principal components before any training for LMNN classification. 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) input space. The square matrix in LMNN-S was of size 350×350, and for dimensionality reduction, the data was projected onto the r leading eigenvectors of linear transformation $\mathbf{L}$. The rectangular matrix in LMNN-R was of size $r \times 350$, where r varied from 15 to 50. The speed-up from ball trees is shown at the top of the graph. The amount of speed-up depends significantly on the amount of dimensionality reduction, but very little on the particular method of dimensionality reduction.

The results show that LMNN can be used effectively for dimensionality reduction. For example, LMNN-R achieves a kNN test error rate of 2.38% in 15 dimensions, only slightly higher than the baseline error rate of 2.33% in the original input space. In this space, moreover, ball trees yield a 15x speedup over baseline kNN search. In 25 dimensions, the LMNN-R error rate drops further to 1.76% while still yielding a 5.7x speed-up. Of the three methods compared in Fig. 13, LMNN-R is the most effective. In fact, though working in many fewer dimensions, LMNN-R obtains results very close to the best results reported in section 4. It is interesting that LMNN-R outperforms LMNN-S, though (as expected) their results converge as the rectangular matrix in LMNN-R becomes more

square. These results show that aggressive dimensionality reduction can be combined with highly accurate *k*NN classification.

7. Discussion

In this paper, we have introduced a new framework for large margin nearest neighbor (LMNN) classification. From labeled training examples, we have shown how to learn a Mahalanobis distance metric for *k*NN classification. The required optimization was formulated as an instance of semidefinite programming. Our framework makes no parametric assumptions about the structure or distribution of the data and scales naturally to problems with large number of classes. On multiple data sets, we have demonstrated that we can significantly improve the accuracy of *k*NN classification by learning a metric in this way. We have also shown that an alternative energy-based decision rule typically leads to further improvements over traditional *k*NN classification.

Beyond the basic framework for LMNN classification, we described several useful and complementary extensions. These included: iterative re-estimation of target neighbor assignments, globally integrated learning of multiple locally linear metrics, kernel methods for LMNN classification, low-rank distance metrics for dimensionality reduction, and ball trees for more efficient gradient computations (in training) and *k*NN search (in testing). These extensions can be adapted and combined to meet the demands of particular applications. For example, to build a highly accurate classifier without regard to the actual computation at test time, our results suggest to train multiple locally linear metrics. At the other extreme, to build a *k*NN classifier that is as fast as possible at test time, our results suggest to combine low-rank distance metrics with ball trees.

Taken as a whole, our results demonstrate the promise and widespread applicability of LMNN classification. Perhaps the greatest promise lies in problems with very large numbers of classes, such as face and identity recognition. The number of classes in these problems can be in the hundreds, thousands, or more. Nearest neighbor methods handle this regime more transparently than other leading methods, such as SVMs. The ideas behind LMNN classification have also been extended by others in various ways (Torresani and Lee, 2007; Kumar et al., 2007). In the appendix, we describe a simple solver that scales well to problems with tens of thousands of examples. A MATLAB implementation of the algorithm is also freely available with this paper.

Future work will concentrate on several open problems. The improved performance with multiple metrics suggests that LMNN classification could benefit from even more adaptive transformations of the input space. It would also be useful to study LMMN classification in the semi-supervised, transductive setting, where only a few labeled inputs are available for training but the unlabeled test set is known in advance. Finally, for many real-world applications in computer vision and information retrieval, the data sets can be much larger than the ones we have studied. For very large data sets, our current implementation for LMNN does not scale as well as simpler eigenvector methods such as PCA, LDA, and RCA. It remains an interesting challenge to scale LMNN to even larger data sets with millions or more training examples.

Acknowledgments

We especially thank John C. Blitzer for his many suggestions to improve the algorithm and his generous help with various data sets. We also thank Koby Crammer for many useful comments and suggestions. This work was supported by NSF Award 0238323.

Appendix A. Solver

We implemented our own special-purpose solver for large-scale problems in LMNN classification. Our solver was designed to exploit the particular structure of the cost function in Eq. (13). The solver iteratively re-estimates the Mahalanobis distance metric as it attempts to minimize the cost 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 implements an iterative sub-gradient projection method to optimize Eq. (14) in terms of the positive semidefinite matrix $\mathbf{M}$. We refer to the Mahalanobis distance metric at the tth iteration as $\mathbf{M}_t$ and to its squared Mahalanobis distance in Eq. (3) as $\mathcal{D}_t$. At each iteration, the optimization takes a step along the sub-gradient to reduce the loss function and then projects $\mathbf{M}_t$ onto the feasible set. In our case, the feasible set is the cone of all positive semidefinite matrices $\mathcal{S}_+$. The following sections derive the gradient and describe the projection onto $\mathcal{S}_+$.

It is worth emphasizing that although we can phrase the optimization of Eq. (14) as a semidefinite program (by introducing nonnegative slack variables to model the hinge loss), in practice our large-scale solver works directly to minimize Eq. (14). The hinge losses that appear in this loss function are not differentiable at all points. Nevertheless, because the loss function is convex, we can compute its sub-gradient and use standard hill-climbing algorithms to find its minimum. It has been shown that such sub-gradient methods converge to the correct solution, provided that the gradient step-size is sufficiently small (Boyd and Vandenberghe, 2004).

A.1 Gradient Computation

The gradient computation can be done most efficiently by careful book-keeping from one iteration to the next. As simplifying notation, let $\mathbf{C}_{ij} = (\vec{x}_i - \vec{x}_j)(\vec{x}_i - \vec{x}_j)^\top$. It is straightforward to express the distances, as defined in Eq. (3), in terms of this notation. In particular, at the tth iteration, we have $\mathcal{D}_t(\vec{x}_i, \vec{x}_j) = \mathrm{tr}(\mathbf{M}_t \mathbf{C}_{ij})$. Consequently, we can rewrite the loss function in Eq. (14) as:

$$\varepsilon(\mathbf{M}_t) \;=\; (1-\mu) \sum_{i,\, j \rightsquigarrow i} \mathrm{tr}(\mathbf{M}_t \mathbf{C}_{ij}) + \mu \sum_{j \rightsquigarrow i,\, l} (1 - y_{il}) \left[1 + \mathrm{tr}(\mathbf{M}_t \mathbf{C}_{ij}) - \mathrm{tr}(\mathbf{M}_t \mathbf{C}_{il})\right]_+ \tag{18}$$

Note that Eq. (18) is piecewise linear with respect to $\mathbf{M}_t$. Let us define a set of triples $\mathcal{N}^t$, such that $(i, j, l) \in \mathcal{N}^t$ if and only if the indices (i, j, l) trigger the hinge loss in the second part of Eq. (18). With this definition, we can write the gradient $\mathbf{G}_t$ of $\varepsilon(\mathbf{M}_t)$ as:

$$\mathbf{G}_t = \frac{\partial \varepsilon}{\mathbf{M}_t} = (1-\mu) \sum_{i,\, j \rightsquigarrow i} \mathbf{C}_{ij} + \mu \sum_{(i,j,l) \in \mathcal{N}^t} (\mathbf{C}_{ij} - \mathbf{C}_{il})\,.$$

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 very 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 $\mathcal{N}_{t+1}$. We can use this fact to derive an extremely efficient update that relates the gradient $\mathbf{G}_{t+1}$ at iteration $t+1$ from the gradient $\mathbf{G}_t$ at iteration t. The update simply subtracts the contributions from triples that are no longer active and adds the contributions of those that just became active:

$$\mathbf{G}_{t+1} = \mathbf{G}_t - \mu \sum_{(i,j,l)\in\mathcal{N}_t-\mathcal{N}_{t+1}} (\mathbf{C}_{ij} - \mathbf{C}_{il}) + \mu \sum_{(i,j,l)\in\mathcal{N}_{t+1}-\mathcal{N}_t} (\mathbf{C}_{ij} - \mathbf{C}_{il}). \tag{19}$$

For small gradient step sizes, the set $\mathcal{N}_t$ changes very little from one iteration to the next. In this case, computing the right hand side of Eq. (19) is extremely fast.

To accelerate the solver even further, we adopt an active set method. 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 exploit the fact 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. We initialize the training procedure by checking all triples and maintaining an active list of those with margin violations; however, a full re-check is only made every 10-20 iterations, depending on fluctuations of the set $\mathcal{N}_t$. For intermediate iterations, we only check for margin violations from among those active triples accumulated over previous iterations. 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 restarts with the newly expanded active set.

A.2 Projection

The minimization of Eq. (18) must enforce the constraint that the matrix $\mathbf{M}_t$ remains positive semidefinite. To enforce this constraint, we project $\mathbf{M}_t$ onto the cone of all positive semidefinite matrices $\mathcal{S}_+$ after each gradient step. This projection is computed from the diagonalization of $\mathbf{M}_t$. Let $\mathbf{M}_t = \mathbf{V}\Delta\mathbf{V}^\top$ denote the eigendecomposition of $\mathbf{M}_t$, where $\mathbf{V}$ is the orthonormal matrix of eigenvectors and Δ is the diagonal matrix of corresponding eigenvalues. We can further decompose $\Delta = \Delta^- + \Delta^+$, where $\Delta^+ = \max(\Delta, 0)$ contains all the positive eigenvalues and $\Delta^- = \min(\Delta, 0)$ contains all the negative eigenvalues. The projection of $\mathbf{M}_t$ onto the cone of positive semidefinite matrices is given by:

$$\mathcal{P}_{\mathcal{S}}(\mathbf{M}_t) = \mathbf{V}\Delta^+\mathbf{V}^\top. \tag{20}$$

The projection effectively truncates any negative eigenvalues from the gradient step, setting them equal to zero.

A.3 Algorithm

Our gradient projection algorithm combined the update rules for the gradient in Eq. (19) and the projection in Eq. (20). A simplified pseudo-code implementation is shown in Algorithm 1. We denote the gradient step-size by $\alpha > 0$. In practice, it worked best to start with a small value of

α. Then, at each iteration, we increased α by a factor of 1.01 if the loss function decreased and decreased α by a factor of 0.5 if the loss function increased.

Algorithm 1 A simple gradient projection pseudo-code implementation.

$\mathbf{M}_0 := \mathbf{I}$ {Initialize with the identity matrix}
$t := 0$ {Initialize counter}
$\mathcal{N}^{(0)}, \mathcal{N}_0 := \{\}$ {Initialize active sets}
$\mathbf{G}_0 := (1-\mu)\sum_{i,j\rightsquigarrow i} \mathbf{C}_{ij}$ {Initialize gradient}
while (not converged) **do**
  if $\mathrm{mod}(t, someconstant) = 0 \ \vee$ (almost converged) {we used *someconstant*=10} **then**
    compute $\mathcal{N}_{t+1}$ exactly
    $\mathcal{N}^{(t+1)} := \mathcal{N}^{(t)} \cup \mathcal{N}_{t+1}$ {Update active set}
  else
    compute $\mathcal{N}_{t+1} \approx \mathcal{N}_{t+1} \cap \mathcal{N}^{(t)}$ { Only search active set}
    $\mathcal{N}^{(t+1)} := \mathcal{N}^{(t)}$ {Keep active set untouched}
  end if
  $\mathbf{G}_{t+1} := \mathbf{G}_t - \mu \sum_{(i,j,l)\in\mathcal{N}_t - \mathcal{N}_{t+1}} (\mathbf{C}_{ij} - \mathbf{C}_{il}) + \mu \sum_{(i,j,l)\in\mathcal{N}_{t+1} - \mathcal{N}_t} (\mathbf{C}_{ij} - \mathbf{C}_{il})$
  $\mathbf{M}_{t+1} := \mathcal{P}_{\mathcal{S}}(\mathbf{M}_t - \alpha \mathbf{G}_{t+1})$ {Take gradient step and project onto SDP cone}
  $t := t+1$
end while
Output $\mathbf{M}_t$

References

A. Bar-Hillel, T. Hertz, N. Shental, and D. Weinshall. Learning a Mahalanobis metric from equivalence constraints. *Journal of Machine Learning Research*, 6(1):937–965, 2006.

S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (4):509–522, 2002.

A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the Twenty Third International Conference on Machine Learning*, pages 97–104, Pittsburgh, PA, 2006.

M. Bilenko, S. Basu, and R.J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 839–846, Banff, Canada, 2004.

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

S. Chopra, R. Hadsell, and Y. LeCun. Learning a similiarty metric discriminatively, with application to face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-05)*, pages 349–356, San Diego, CA, 2005.

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

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

S. Dasgupta and A. Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report 99–006, International Computer Science Institute, UC Berkeley, 1999.

T. De Bie, M. Momma, and N. Cristianini. Efficiently Learning the Metric with Side-Information. *Lecture Notes in Computer Science*, pages 175–189, 2003.

T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. In *Journal of Artificial Intelligence Research*, volume 2, pages 263–286, 1995.

R. A. Fisher. The use of multiple measures in taxonomic problems. *Annals of Eugenics*, 7:179–188, 1936.

J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Advances in Neural Information Processing Systems 18*, 2006.

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

P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, 1986.

M. P. Kumar, P. H. S. Torr, and A. Zisserman. An invariant large margin nearest neighbour classifier. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV-07)*, pages 1–8, Rio de Janeiro, Brazil, 2007.

J.T. Kwok and I.W. Tsang. Learning with idealized kernels. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-03)*, pages 400–407, Washington, D.C., 2003.

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

Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard, and V. Vapnik. A comparison of learning algorithms for handwritten digit recognition. In F. Fogelman and P. Gallinari, editors, *Proceedings of the 1995 International Conference on Artificial Neural Networks (ICANN-95)*, pages 53–60, Paris, 1995.

T. Liu, A. W. Moore, A. Gray, and K. Yang. An investigation of practical approximate nearest neighbor algorithms. In L. K. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 825–832. MIT Press, Cambridge, MA, 2005.

A. Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

K.-R. Müller, S. Mika, G. Räsch, K. Tsuda, and B. Schökopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001.

S. Omohundro. Efficient algorithms with neural network behavior. *Complex Systems*, 1:273–347, 1987.

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

B. Schölkopf, A. J. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

S. Shalev-Shwartz, Y. Singer, and A. Y. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the Twenty First International Conference on Machine Learning (ICML-04)*, pages 94–101, 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.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, pages 888–905, August 2000.

P. Y. Simard, Y. LeCun, and J. Decker. Efficient pattern recognition using a new transformation distance. In S. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems 6*, pages 50–58, San Mateo, CA, 1993. Morgan Kaufman.

L. Torresani and K.C. Lee. Large margin component analysis. In B. Schölkopf, J. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*, pages 1385–1392. MIT Press, Cambridge, MA, 2007.

I.W. Tsang, P.M. Cheung, and J.T. Kwok. Kernel relevant component analysis for distance metric learning. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN-05)*, volume 2, pages 954–959, Montreal, Canada, 2005.

M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1): 71–86, 1991.

L. Vandenberghe and S. P. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, March 1996.

M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision (ICCV-07)*, pages 1–8, 2007.

K. Q. Weinberger and L. K. Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of the Twenty Fifth International Conference on Machine learning*, pages 1160–1167, Helsinki, Finland, 2008.

K. Q. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1473–1480. MIT Press, Cambridge, MA, 2006.

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*, pages 521–528, Cambridge, MA, 2002. MIT Press.