Efficiently Learning a Distance Metric for Large Margin Nearest Neighbor Classification

Kyoungup Park[1,3], **Chunhua Shen**[2,1*], **Zhihui Hao**[4†], **Junae Kim**[1,3]
[1]NICTA[‡] [2]University of Adelaide [3]Australian National University [4]Beijing Institute of Technology

Abstract

We concern the problem of learning a Mahalanobis distance metric for improving nearest neighbor classification. Our work is built upon the large margin nearest neighbor (LMNN) classification framework. Due to the semidefiniteness constraint in the optimization problem of LMNN, it is not scalable in terms of the dimensionality of the input data. The original LMNN solver partially alleviates this problem by adopting alternating projection methods instead of standard interior-point methods. Still, at each iteration, the computation complexity is at least $O(D^3)$ (D is the dimension of input data). In this work, we propose a column generation based algorithm to solve the LMNN optimization problem much more efficiently. Our algorithm is much more scalable in that at each iteration, it does not need full eigen-decomposition. Instead, we only need to find the leading eigenvalue and its corresponding eigenvector, which is of $O(D^2)$ complexity. Experiments show the efficiency and efficacy of our algorithms.

Introduction

The distance metric learning is an important topic in machine learning and has been successfully integrated with classification and clustering methods, including k-nearest neighbor (kNN) and k-means clustering. The basic idea is to learn a metric with which distances between examples belonging to the same class are minimized, while distances between different classes are maximized. For instance, given a bunch of points $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, we are interesting in designing a quadratic Mahalanobis distance $\mathrm{dist}_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_{\mathbf{M}} = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{M}(\boldsymbol{x}_i - \boldsymbol{x}_j)}$ with $\mathbf{M} \succcurlyeq 0$. Here $\mathbf{M}$ is a positive semidefinite (p.s.d) matrix. Therefore, a constrained semidefinite programming (SDP) is usually involved, which makes it a difficult problem to solve.

*C. Shen's research was supported in part by the Australian Research Council through its special research initiative in bionic vision science and technology grant to Bionic Vision Australia.

†Z. Hao's contribution was made when visiting NICTA Canberra Research Laboratory.

‡NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Center of Excellence program.

Since the first metric learning algorithm based on SDP was proposed by (Xing et al. 2002) for learning a Mahalanobis metric for unsupervised clustering, much research interest has been focused on distance metric learning, including Neighborhood Component Analysis (NCA) (Goldberger et al. 2004), Maximally Collapsing Metric Learning (MCML) (Globerson and Roweis 2005), Large Margin Nearest Neighbor (LMNN) (Weinberger, Blitzer, and Saul 2005), and Positive Semidefinite Boosting (PSDBoost) (Shen, Welsh, and Wang 2008).

The work presented here is mainly motivated by the work of LMNN and PSDBoost. The idea of LMNN is to maximize the margin between different classes, while keeping the distances between same-class instances as minimum as possible. LMNN has been reported the state-of-the-art classification performance (Weinberger, Blitzer, and Saul 2005). However, a drawback of LMNN is that it is not scalable in terms of dimensionality of the training data due to the semidefinite-ness constraint. The original implementation of LMNN partially remedies this problem by employing an alternating projection method instead of using conventional interior-point methods. However, the computation complexity is still high: at each iteration, the complexity is at least $O(D^3)$ with D being the dimension of the input data. At each iteration, a full eigen-decomposition is needed to project the intermediate solution to the positive semidefinite cone. Another work that is similar to ours is PSDBoost (Shen, Welsh, and Wang 2008), where the SDP problem involved in metric learning is converted into an additive Linear Programming (LP) problem based on the observation that any positive semidefinite matrix can be decomposed into a sum of linear positive combination of trace-one rank-one matrices. Although a simplified version of PSDBoost is proposed later—BoostMetric in (Shen et al. 2009)—which performs a stage-wise learning procedure for minimizing an exponential loss function, PSDBoost is still advantageous in terms of the convergence speed due to the totally corrective property, and flexible in optimizing any type of loss functions. We implement our algorithm in this work using both the exponential loss and logistic loss. Note that both PSDBoost and BoostMetric solve a simplified version of the original LMNN optimization problem in the sense that they ignore the within-class distance information. In contrast to the within-class distance as a regularization term in LMNN,

PSDBoost or BoostMetric simply bounds the trace of $\mathbf{M}$ as regularization. It remains unclear if the matrix generation technique in PSDBoost or BoostMetric can be used to solve the original optimization problem of LMNN. We give an affirmative answer to this question in this work.

In this work, we propose a column generation based algorithm, similar to PSDBoost and BoostMetric. It is much more scalable in high-dimensional data sets since it avoids the full eigen-decomposition. However, unlike PSDBoost which only takes the distances between different classes into consideration, we simultaneously optimize the intra-class and inter-class distances in this paper. Therefore, it can be viewed as a general extension of PSDBoost to some extent. We also introduce the Stochastic Gradient Descent (SGD) to enable our algorithm on large-scale data sets. SGD is simple to implement and has shown convincing performance for coping with large datasets and online learning (Bottou and Bousquet 2008; Shalev-Shwartz and Srebro 2008).

Algorithms

In this section, firstly, we review the LMNN method and then propose our algorithm based on the column generation technique.

Let $\boldsymbol{x}_i \in \mathbb{R}^D, i = 1, 2, \ldots, n$ denote the training samples. D is the dimensionality of the input feature vector and n is the total number of the data set. The Mahalanobis distance can be viewed as a linear projection with matrix $\mathbf{L} \in \mathbb{R}^{D\times d}$ followed by calculating the Euclidean distance. For points $(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the Mahalanobis distance is defined as

$$\mathrm{dist}_{ij}^2 = \left\|\mathbf{L}^\top \boldsymbol{x}_i - \mathbf{L}^\top \boldsymbol{x}_j\right\|_2^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{L}\mathbf{L}^\top (\boldsymbol{x}_i - \boldsymbol{x}_j).$$

If we directly work on $\mathbf{L}$, the pair-wise distance comparison will result in non-convex constraints and thus the overall problem is non-convex (the difference of quadratic terms in $\mathbf{L}$ is non-convex). Therefore, we instead try to learn the matrix $\mathbf{M} = \mathbf{L}\mathbf{L}^\top$, which changes the problem into a convex one in $\mathbf{M}$ (Boyd and Vandenberghe 2004). This *linearization* technique has been widely used to convexify a problem in convex optimization.

Given a training data set, a triplet set $\mathbb{S} = \{(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_l)_r \mid \mathrm{dist}_{ij} < \mathrm{dist}_{il}\}$, $r = 1, \ldots, |\mathbb{S}|$ is selected among the k nearest neighbors for each instance $\boldsymbol{x}_i \in \mathbb{R}^D$. Here $\boldsymbol{x}_j$ has the same label with $\boldsymbol{x}_i$, while the $\boldsymbol{x}_l$ has a different one. $|\mathbb{S}|$ denotes the size of the set. The optimization problem defined in LMNN is as below:

$$\begin{aligned} \min \sum_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{M} (\boldsymbol{x}_i - \boldsymbol{x}_j) + C \sum_r \xi_r \qquad \text{(P0)} \\ \text{s.t. } (\boldsymbol{x}_i - \boldsymbol{x}_l)^\top \mathbf{M} (\boldsymbol{x}_i - \boldsymbol{x}_l) - (\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \mathbf{M} (\boldsymbol{x}_i - \boldsymbol{x}_j) \\ \geq 1 - \xi_r, r = 1, \ldots, |\mathbb{S}| \\ \xi_r \geq 0,\ \mathbf{M} \succcurlyeq 0. \end{aligned}$$

Like in Support Vector Machines (SVM) (Cristianini and Shawe-Taylor 2000), slack variables ξ_r are introduced to implement *soft* margins. To simplify exposition, denote

$$\mathbf{A} = \sum_{ij} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top, \tag{1}$$

$$\mathbf{B}_r = (\boldsymbol{x}_i - \boldsymbol{x}_l)(\boldsymbol{x}_i - \boldsymbol{x}_l)^\top - (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top. \tag{2}$$

Then we can rewrite the problem as

$$\begin{aligned} \min \langle \mathbf{M}, \mathbf{A} \rangle + C \sum_r \xi_r \qquad \text{(P1)} \\ \text{s.t. } \langle \mathbf{M}, \mathbf{B}_r \rangle \geq 1 - \xi_r, \forall r, \\ \xi_r \geq 0, \\ \mathbf{M} \succcurlyeq 0. \end{aligned}$$

Note that any positive semidefinite matrix can be decomposed into a sum of linear positive combination of trace-one rank-one matrices, *i.e.* $\mathbf{M} = \sum_{t=1}^T w_t \mathbf{Z}_t$ with $w_t \geq 0, \mathbf{rank}(\mathbf{Z}_t) = 1$ and $\mathbf{Tr}(\mathbf{Z}_t) = 1$, for all $t = 1, \ldots, T$ (Shen, Welsh, and Wang 2008). Therefore we have

$$\langle \mathbf{M}, \mathbf{A} \rangle = \sum_t w_t \langle \mathbf{Z}_t, \mathbf{A} \rangle = \sum_t w_t a_t$$

$$\langle \mathbf{M}, \mathbf{B}_r \rangle = \sum_t w_t \langle \mathbf{Z}_t, \mathbf{B}_r \rangle = \sum_t w_t \mathbf{H}_{rt} = \mathbf{H}_{r:}\boldsymbol{w},$$

then the optimization program becomes

$$\begin{aligned} \min \sum_t w_t a_t + C \sum_r \xi_r \qquad \text{(P2)} \\ \text{s.t. } \sum_t w_t \mathbf{H}_{rt} \geq 1 - \xi_r,\ \xi_r \geq 0, \forall r;\ w_t \geq 0. \end{aligned}$$

Here we have introduced a new matrix $\mathbf{H}$ with $\mathbf{H}_{rt} = \langle \mathbf{Z}_t, \mathbf{B}_r \rangle$; and $\mathbf{H}_{r:}$ is the r-th row. PSDBoost does not have the term of $\langle \mathbf{M}, \mathbf{A} \rangle$, which measures the intra-class distances of input data. Instead, PSDBoost essentially sets $\mathbf{A}$ to be an identity matrix. In this sense, our algorithm solves a more general problem than the former.

Learning with the Exponential Loss

Different loss functions have been explored in boosting research (Schapire 1999) to design distinctive algorithms, which motivates us to replace the hinge loss in (P2) with a smooth loss function, for example, the exponential loss.

Let us define the margin $\rho_r = \mathrm{dist}_{il}^2 - \mathrm{dist}_{ij}^2$. Then the problem with exponential loss is

$$\begin{aligned} \min \upsilon \sum_t w_t a_t + \log\left(\sum_r \exp(-\rho_r)\right) \qquad \text{(P2.1)} \\ \text{s.t. } \boldsymbol{w} \geq 0,\ \rho_r = \mathbf{H}_{r:}\boldsymbol{w},\ \forall r, \end{aligned}$$

where $\upsilon = 1/C$ is the regularization coefficient. Note that (i) we solve the logarithmic version of the sum of exponential loss instead, which does not change the optimization problem since the logarithm function is monotonically decreasing. Still, this is a convex problem; (ii) The auxiliary variables ρ_r are introduced for deriving a Lagrange dual problem, which will become clear later.

The Lagrangian of the problem (P2.1) is defined as the augmented loss function:

$$\begin{aligned} L(\boldsymbol{\rho}, \boldsymbol{w}, \boldsymbol{u}, \boldsymbol{p}) = \upsilon \sum_t w_t a_t + \log\left(\sum_r \exp(-\rho_r)\right) \\ + \sum_r u_r (\rho_r - \sum_t w_t \mathbf{H}_{rt}) - \sum_t p_t w_t \end{aligned}$$

where $\boldsymbol{u}$ and $\boldsymbol{p} \geq 0$ are Lagrange multipliers. The dual problem is defined as the *infimum* of the Lagrangian:

$$\inf_{\boldsymbol{\rho},\boldsymbol{w}} L = \inf_{\boldsymbol{\rho}} \left(\log \left(\sum_r \exp(-\rho_r) \right) + \sum_r u_r \rho_r \right) + \inf_{\boldsymbol{w}} \left(\upsilon \boldsymbol{a}^\top - \sum_r u_r \mathbf{H}_{r:} - \boldsymbol{p}^\top \right) \boldsymbol{w}.$$

Considering that the partial derivative of the above equation on w_t vanishes at optimal value, *i.e.* $\frac{\partial L}{\partial w_t} = 0$ and applying $\boldsymbol{p} \geq 0$, we have

$$\upsilon a_t \geq \sum_r u_r \mathbf{H}_{r:}. \tag{3}$$

Since the Fenchel conjugate of log-sum-exp function is the negative entropy function (Boyd and Vandenberghe 2004), the Lagrange dual problem of (P2.1) can be written as

$$\begin{aligned} \max_{\boldsymbol{u}} & -\sum_r u_r \log u_r && \text{(D2.1)} \\ \text{s.t.} & \sum_r u_r \mathbf{H}_{r:} \leq \upsilon a_t, \\ & \sum_r u_r = 1,\ u_r \geq 0. \end{aligned}$$

Due to the strong duality, the dual gap should be zero, which means that the optimal value of the primal and the dual problem should be the same. We can establish the link between the primal and dual variables at optimality by the KKT conditions:

$$u_r^* = \frac{\exp(-\rho_r^*)}{\sum_r \exp(-\rho_r^*)}. \tag{4}$$

Clearly, given the primal value, we can obtain the dual variable u_r, $r = 1, \ldots |\mathbb{S}|$.

Learning with the Logistic Loss

Next we apply the logistic loss to the original primal problem (P2), and then the problem turns to be

$$\begin{aligned} \min\ & \upsilon \sum_t w_t a_t + \sum_r \log\left(1 + \exp(-\rho_r)\right) && \text{(P2.2)} \\ \text{s.t.}\ & \boldsymbol{w} \geq 0 \\ & \rho_r = \sum_t w_t \mathbf{H}_{rt} = \mathbf{H}_{r:}\boldsymbol{w}\ \forall r \end{aligned}$$

where $\upsilon = 1/C$. It is easy to derive the dual problem of (P2.2) as in the case of exponential loss. The Lagrange dual problem can be written as

$$\begin{aligned} \max_{\boldsymbol{u}} & -\sum_r \left((1-u_r)\log(1-u_r) + u_r \log u_r\right) && \text{(D2.2)} \\ \text{s.t.}\ & 0 < u_r < 1,\ \sum_r u_r \mathbf{H}_{r:} \leq \upsilon a_t. \end{aligned}$$

The KKT conditions in this case link the optimal values as below:

$$u_r^* = \frac{\exp(-\rho_r^*)}{1 + \exp(-\rho_r^*)}. \tag{5}$$

Algorithm 1 Large margin nearest neighbor learning using column generation

Input:

- Triplets $(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k) \in \mathbb{S}$
- Set T for the maximum number of iterations.
- Set the regularization parameter υ.

Initialize:
  $u_r^0 = \frac{1}{|\mathbb{S}|}, r = 1 \cdots |\mathbb{S}|$
  Compute $\mathbf{A}$ and $\mathbf{B}_r, r = 1, 2, \cdots,$ using (1) and (2).
for $t = 1, 2, \cdots, T$ **do**
  Find a new base $\mathbf{Z}_t$ by finding $\lambda_{max}(\hat{\mathbf{A}})$ and the eigenvector in (7);
  if $\lambda_{max}(\hat{\mathbf{A}}) < 0$ **then**
    break (converged);
  Add the matrix $\mathbf{Z}_t$ to the problem (P2.1) or (P2.2);
  Update $\boldsymbol{w}$ by solving the primal problem, using tools like L-BFGS-B;
  Update $\boldsymbol{u}$ in (4);

Output: The p.s.d. matrix $\mathbf{M} \in \mathbb{R}^{D \times D}$, $\mathbf{M} = \sum_{t=1}^{T} w_t \mathbf{Z}_t$.

Column Generation Based Optimization

Obviously, if all the base matrix $\mathbf{Z}_t$ are known, we can solve the primal problem easily by any optimization tool like L-BFGS-B. However, the number of the candidates may be infinite and can not be accessed at once. To overcome this difficulty, we use the column generation (CG) technique (Lübbecke and Desrosiers 2005), which helps us to solve the problem by finding the most violated constraint at each iteration and adding them one by one until the optimization problem converges.

So here, we need to solve the following problem at each iteration

$$\begin{aligned} \mathbf{Z}^* &= \text{argmax}_{\mathbf{Z}} \sum_r u_r \mathbf{H}_{r:} - \upsilon a_t && \text{(6)} \\ &= \text{argmax}_{\mathbf{Z}} \left\langle \sum_r u_r \mathbf{B}_r - \upsilon \mathbf{A}, \mathbf{Z}_t \right\rangle. \end{aligned}$$

Now, we can see that the above optimization problem (6) can be efficiently solved by eigenvalue-decomposition (EVD). If we introduce a new symbol

$$\hat{\mathbf{A}} = \sum_r u_r \mathbf{B}_r - \upsilon \mathbf{A}, \tag{7}$$

it is clear that the largest eigenvalue of $\hat{\mathbf{A}}$, $\lambda_{max}(\hat{\mathbf{A}})$, and its corresponding eigenvector ξ_1 gives the solution to the above problem. Also, $\lambda_{max}(\hat{\mathbf{A}})$ can be used as a stop criteria. If $\lambda_{max}(\hat{\mathbf{A}}) < 0$, it means that we cannot find any violated base matrix and the optimization problem converges. Algorithm 1 summarizes the entire learning process.

More Efficient Stochastic Gradient Descent

Here we introduce the stochastic gradient descent (SGD) method to speed up the computation in updating $\boldsymbol{w}$ when

Dataset	# train	# test	dim.	# classes	# runs
bal	439	186	4	3	5
synthetic control	420	180	60	6	5
wine	126	52	13	3	5
german.numer	700	300	24	2	5
splice	701	299	60	2	5
connect-4	701	299	126	2	5
CTG	1493	633	21	10	5
segment	1617	693	19	7	5
spambase	3222	1379	57	2	5
mushrooms	5688	2436	112	2	5
usps	7700	3300	256	10	5
letter	14015	5985	16	26	5
mnist (PCA)	60000	10000	300	10	5
optdigit	3823	1797	64	10	1
mnist	60000	10000	784	10	1

Table 1: Properties of Data Sets

solving the primal problem. Stochastic gradient descent is an optimization method to minimize an objective function which is a sum of a set of differentiable functions. Clearly in our case, both primal problems belong to this category.

Because the triple set can be very large, to assess the objective function may result in expensive evaluations of the gradients from all summand functions. If a huge training set is given but simple formulas are not available for that case, evaluating the sums of gradients costs a lot because computing the gradient demands the evaluation of all the summand functions' gradients. To decrease the computational complexity at each iteration, stochastic gradient descent takes samples of a subset of summand functions at each iteration, which is intuitive.

There are many versions of stochastic gradient methods, according to the different sampling techniques and different sizes of subsets. We simply apply the idea by uniformly randomly generating the triplets (Line 8 in Algorithm 1). Note that in some cases, the summand functions may be of a simple form, which enables inexpensive evaluations of the sum-function and the sum-gradient. In these cases, stochastic sampling may not help to reduce the computational time.

More sophisticated methods in active learning may be applied here to achieve even more efficiency. We leave this as a future topic.

Experiments

We performed experiments using various data sets from UCI repository[1] and LIBSVM[2]. We have all data sets scaled between -1 and 1, similar to the preprocessing in LIBSVM data sets. Then, train/test sets are randomly shuffled and divided into 70/30 except for the case a pre-split is provided (mnist and optdigit). We generated the triplets from the training parts in the same way as (Weinberger, Blitzer, and Saul 2005). For each instance, 3 nearest neighbors with same and different labels were grouped respectively. All the experiments were run for 5 times per data set. The detailed experiment conditions are summarized in the Table 1.

[1]http://archive.ics.uci.edu/ml/datasets.html

[2]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

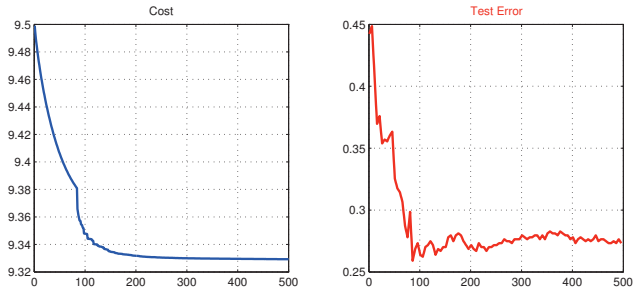

Figure 1: Comparison of the objective values and test error rates in the iteration number. The objective value converges fast and becomes stable quickly.

We use the original LMNN source code from the authors' website. We let LMNN update the matrix M ($\mathbf{M} = \mathbf{L}\mathbf{L}^\top$) for global optimization. For our methods, we simply set the parameter υ as small as 10^{-7} and the maximum iteration $T = 500$. We run our method on one of the data sets and plot the results in Figure 1. As shown in Figure 1, our method converges very fast but can experience so-called *tailing-off effect*, which is also mentioned in (Shen, Welsh, and Wang 2008). An approximate convergence is already obtained at the early iterations and the remaining iterations seem useless to the final accuracy. Therefore, we may be able to stop the optimization earlier to obtain an acceptable accuracy. For this reason, we calculate the relative difference of the objective function at neighboring iterations and stop the optimization using this criterion.

Table 2 shows the test errors (%) of different loss functions with SGD or deterministic gradient descent. The baseline algorithm is the 'Euclidean' which uses the standard Euclidean distance for the kNN classifier. As aforementioned, LMNN uses the Hinge loss function and our methods use the exponential and the logistic loss functions respectively. Again, we uniformly randomly sample 50% of the triplets to calculate the gradient in the case of both exponential and logistic loss functions. As shown in the table, our results achieve similar test errors to LMNN and the results are persistently better than those of Euclidean.

Table 3 compares the computational time of our methods and LMNN. The experiments were performed on a workstation with an Intel Xeon E5520 CPU (2.27GHz) and 32GB RAM (only single core is used). Our code is implemented in MATLAB. We have used L-BFGS-B as the gradient descent optimization tool for our methods. As can be seen from the table, our algorithms are consistently faster than LMNN. Another observation is that random sampling may not be always faster than the standard gradient descent due to the fact that random sampling may need more iterations. In general, random sampling is faster than the deterministic counterpart.

Our next experiment is to compare the computational time in different dimensions and sample sizes. We use a toy data generated artificially. For the test against the input dimension, we keep the sample size of the training data to be 500. For the test against sample size, we fix the input dimen-

Dataset	Euclidean	Hinge (LMNN)	Exponential	Exp. (Rand)	Logistic	Logistic (Rand)
bal	18.39 (1.63)	17.74 (2.01)	10.11 (3.03)	9.68 (2.87)	9.68 (2.77)	10.22 (2.63)
synthetic control	2.22 (0.56)	0.44 (0.46)	0.89 (0.84)	0.56 (0.96)	0.89 (0.50)	0.44 (0.46)
wine	4.23 (2.11)	2.69 (2.19)	2.31 (1.61)	0.77 (1.05)	2.31 (2.11)	3.08 (1.72)
german.numer	31.07 (1.38)	31.00 (1.58)	28.87 (2.85)	29.4 (3.03)	29.87 (0.80)	30.47 (2.14)
splice	30.5 (0.44)	18.93 (2.44)	19.73 (2.26)	19.33 (2.26)	20 (1.65)	19.53 (0.51)
connect-4	31.04 (1.56)	19.67 (2.47)	19.67 (3.07)	19.53 (2.12)	19.06 (2.91)	20 (2.62)
CTG	23.51 (1.39)	21.42 (0.68)	21.64 (1.16)	20.76 (1.12)	20.73 (0.89)	20.98 (1.14)
segment	3.92 (0.47)	3.61 (0.72)	3.29 (0.66)	3.32 (0.57)	3.35 (0.66)	3.35 (0.75)
spambase	10.11 (0.34)	7.9 (0.39)	8.77 (0.26)	8.17 (0.45)	7.47 (0.75)	7.24 (0.62)
mushrooms	0 (0)	0.27 (0.28)	0.26 (0.07)	0.35 (0.38)	0.05 (0.13)	0.01 (0.03)
usps	4.88 (0.26)	2.41 (0.22)	2.52 (0.27)	2.38 (0.16)	2.6 (0.22)	2.58 (0.17)
letter	4.62 (0.13)	3.60 (0.09)	3.18 (0.25)	3.25 (0.35)	2.79 (0.17)	2.8 (0.18)
mnist (PCA)	2.37 (0.13)	1.76 (0.15)	1.80 (0.15)	1.76 (0.13)	2.29 (0.23)	2.28 (0.24)
optdigit	2.11 (-)	1.67 (-)	1.56 (-)	1.78 (-)	1.67 (-)	1.56 (-)
mnist	2.83 (-)	2.12 (-)	2.2 (-)	2.14 (-)	2.75 (-)	2.74 (-)

Table 2: Test error rates (%) performed on some UCI and LIBSVM data sets. Mean and standard deviation are reported here. "Rand" means 50% random sampling of triplets. Our methods with the exponential and logistic loss functions show very similar classification accuracy as LMNN.

Dataset	Hinge (LMNN)	Exponential	Exponential (Rand)	Ratio	Logistic	Logistic (Rand)	Ratio
bal	6.54 (0.29)	**0.2 (0.13)**	0.33 (0.14)	31.96	**0.22 (0.14)**	0.33 (0.29)	29.53
synthetic control	9.45 (2.01)	**0.78 (0.10)**	1.86 (0.20)	12.04	**0.95 (0.13)**	1.92 (0.71)	9.93
wine	2.00 (0.31)	**0.1 (0.01)**	0.19 (0.03)	19.11	**0.15 (0.02)**	0.26 (0.02)	13.25
german.numer	4.57 (0.41)	0.72 (0.20)	**0.65 (0.07)**	7.04	**0.36 (0.00)**	0.45 (0.08)	12.71
splice	16.85 (2.26)	**8.56 (0.80)**	10.09 (1.61)	1.97	**1.92 (0.50)**	4.28 (0.69)	8.79
connect-4	26.1 (11.82)	24.13 (3.23)	**5.83 (0.51)**	4.48	**1.81 (0.29)**	4.63 (1.30)	14.44
CTG	34.36 (5.30)	**1.06 (0.07)**	1.55 (0.12)	32.40	**0.87 (0.04)**	0.95 (0.10)	39.55
segment	28.15 (5.81)	3.46 (0.48)	**2.71 (0.99)**	10.37	**1.18 (0.14)**	1.43 (0.06)	23.84
spambase	1912.7 (962.90)	22.38 (11.07)	**8.05 (0.70)**	237.62	**17.24 (7.91)**	19.39 (8.81)	110.91
mushrooms	192.88 (307.95)	**35.15 (4.82)**	38.45 (3.71)	5.49	**35.18 (2.46)**	75.28 (28.02)	5.48
usps	6636.9 (162.77)	3122.8 (294.04)	**2351.7 (312.88)**	2.82	**481.01 (25.98)**	581.87 (60.46)	13.80
letter	1206.3 (304.94)	**40.74 (1.59)**	53.56 (7.83)	29.61	**39.44 (0.19)**	43.69 (1.98)	30.59
mnist (PCA)	50394 (2222.9)	10206 (2570.9)	**8851.3 (444.78)**	5.69	**7228.6 (39.35)**	7424 (41.55)	6.97
optdigit	198.24 (-)	102.27 (-)	**40.79 (-)**	4.86	110.25 (-)	**29.45 (-)**	6.73
mnist	380760 (-)	**183230 (-)**	196747 (-)	2.08	187650 (-)	**73814 (-)**	5.16

Table 3: Comparison of computational time in seconds. Mean and standard deviation are reported here. "Rand" means 50% random sampling of triplets. "Ratio" calculates the speed ratio between LMNN and our algorithm (the relatively faster one, although there is no significant difference between our deterministic and stochastic versions). Our methods in general are much faster than LMNN.

sion to be 50. The results are averaged over 5 runs. Figure 2 clearly shows the difference between LMNN and our algorithms. As expected, the computation time grows with the dimension and the sample size. We can see that our methods increases much more slowly than LMNN, which means that ours are more scalable on large-size data sets.

Influence of υ For our algorithms, we do not need careful cross-validation for the regularization parameter υ. We have simply set the parameter υ to a small value. We show here that this regularization parameter does not have much impact on the final accuracy as long as we keep it small. The parameter υ controls the cost of the distance between same-labeled instances, which may already be reflected on the second part in the cost function. We varied the parameter value from 10^{-8} to 10^{-5} and ran 5 times with three data sets. See Table 4 for details. Experiment results verify our conjecture that indeed υ is not important as long as it is a small value.

In the last experiment, we want to check the results with different sampling rate in SGD. In the above experiments, we have set the sampling rate to be 50%. Unlike general optimization techniques, the SGD method does not depend much on the number of samples (Bottou and Bousquet 2008). Table 5 reports the empirical results with three data sets. We changed sampling rates from 10% to 70%. The difference between the results is not significant in terms of test accuracy and computation time. It seems that the optimal

Dataset	10^{-8}	10^{-7}	10^{-6}	10^{-5}
syn.crtl	0.67 (0.91)	0.89 (0.75)	0.56 (0.56)	0.78 (0.84)
wine	1.92 (1.36)	1.92 (2.36)	1.92 (1.92)	1.92 (1.92)
connect-4	18.73 (2.02)	19.46 (2.56)	19.20 (3.37)	18.33 (2.50)

Table 4: Test error rates (%) of 3-NN with different values of the parameter υ, with the exponential loss and SGD. We can see that our methods are not sensitive to υ. This is consistent to the finding in (Shen et al. 2009).

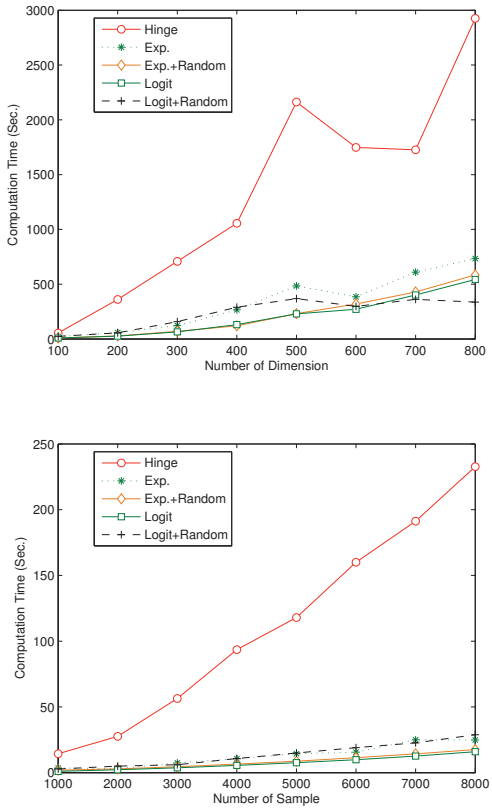

Figure 2: Scalability tests in the input dimension and the number of samples. As expected, our algorithms are much more scalable than the original LMNN implementation. In the first figure, LMNN does not always grow with the input dimension. This may be caused by the active constraint generation technique used in the original LMNN implementation.

sampling rate is data dependent. It remains unclear about how to find an optimal sampling rate.

Conclusion

In this work, we have proposed a new technique to solve the large margin nearest neighbor metric learning problem. The proposed algorithms are much simpler to implement than the original LMNN implementation: only eigenvalue decomposition and an off-the-shelf gradient descent tool are needed. In contrast, the alternating projection method employed by the original LMNN algorithm is much more complicated.

We show that the proposed algorithms are much more efficient and scalable than the original LMNN method, with a comparable classification accuracy. Experiments also show that we do not need to cross validate any parameters for our algorithms, which further simplifies the use of our algorithms.

Dataset		10%	30%	50%	70%
syn.crtl	Error	1.11 (0.88)	1.44 (1.50)	0.56 (0.68)	1.11 (0.79)
	Time	2.91 (0.27)	2.96 (0.34)	3.31 (0.43)	3.16 (0.21)
	Iter.	11 (0.71)	12 (2.12)	13.8 (2.38)	12.8 (0.84)
letter	Error	3.20 (0.31)	3.10 (0.23)	3.18 (0.25)	3.14 (0.3)
	Time	85.93 (2.84)	96.37 (8.98)	101.27 (8.57)	79.39 (0.71)
	Iter.	26.8 (4.38)	24.6 (2.97)	28 (3.08)	25.8 (0.84)
connect-4	Error	18.93 (1.73)	19.06 (1.34)	19.2 (2.12)	19.4 (2.48)
	Time	11.38 (2.04)	18.53 (1.48)	32.17 (5.95)	42.43 (4.43)
	Iter.	34 (7.11)	54.2 (4.15)	78.4 (9.24)	93 (9.67)

Table 5: Experiment results with different sampling rates. The training time is in seconds. This experiment was conducted using the exponential loss function with SGD.

References

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

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

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

Globerson, A., and Roweis, S. 2005. Metric learning by collapsing classes. In *Proc. Adv. Neural Inf. Process. Syst.*

Goldberger, J.; Roweis, S.; Hinton, G.; and Salakhutdinov, R. 2004. Neighbourhood component analysis. In *Proc. Adv. Neural Inf. Process. Syst.* MIT Press.

Lübbecke, M. E., and Desrosiers, J. 2005. Selected topics in column generation. *Operation Res.* 53(6):1007–1023.

Schapire, R. E. 1999. Theoretical views of boosting and applications. In *Proc. Int. Conf. Algorithmic Learn. Theory*, 13–25. London, UK: Springer-Verlag.

Shalev-Shwartz, S., and Srebro, N. 2008. Svm optimization: Inverse dependence on training set size. In *Proceedings of the 25th International Conference on Machine Learning*, 928–935.

Shen, C.; Kim, J.; Wang, L.; and van den Hengel, A. 2009. Positive semidefinite metric learning with boosting. In Bengio, Y.; Schuurmans, D.; Lafferty, J.; Williams, C.; and Culotta, A., eds., *Proc. Adv. Neural Inf. Process. Syst.*, 1651–1659. Vancouver, B.C., Canada: MIT Press.

Shen, C.; Welsh, A.; and Wang, L. 2008. PSDBoost: Matrix-generation linear programming for positive semidefinite matrices learning. In Koller, D.; Schuurmans, D.; Bengio, Y.; and Bottou, L., eds., *Proc. Adv. Neural Inf. Process. Syst.*, 1473–1480. Vancouver, B.C., Canada: MIT Press.

Weinberger, K. Q.; Blitzer, J.; and Saul, L. K. 2005. Distance metric learning for large margin nearest neighbor classification. In *Proc. Adv. Neural Inf. Process. Syst.*, 1473–1480.

Xing, E.; Ng, A.; Jordan, M.; and Russell, S. 2002. Distance metric learning, with application to clustering with side-information. In *Proc. Adv. Neural Inf. Process. Syst.* MIT Press.