Scalable Large-Margin Mahalanobis Distance Metric Learning
Chunhua Shen, Junae Kim, and Lei Wang

Abstract—For many machine learning algorithms such as k-nearest neighbor (k-NN) classifiers and k-means clustering, often their success heavily depends on the metric used to calculate distances between different data points. An effective solution for defining such a metric is to learn it from a set of labeled training samples. In this work, we propose a fast and scalable algorithm to learn a Mahalanobis distance metric. The Mahalanobis metric is an extension of the Euclidean distance metric on the input data that have been linearly transformed. By employing the principle of margin maximization to achieve better generalization performances, this algorithm formulates the metric learning as a convex optimization problem and a positive semidefinite (p.s.d.) matrix is the unknown variable. Based on an important theorem that a p.s.d.trace-one matrix can always be represented as a convex combination of multiple rank-one matrices, our algorithm accommodates any differentiable loss function and solves the resulting optimization problem using a specialized gradient descent procedure. During the course of optimization, the proposed algorithm maintains the positive semidefiniteness of the matrix variable that is essential for a Mahalanobis metric. Compared with conventional methods like standard interior-point algorithms [2] or the special solver used in large margin nearest neighbor [24], our algorithm is much more efficient and has a better performance in scalability. Experiments on benchmark data sets suggest that, compared with state-of-the-art metric learning algorithms, our algorithm can achieve a comparable classification accuracy with reduced computational complexity.

Index Terms—Distance metric learning, large-margin nearest neighbor, Mahalanobis distance, semidefinite optimization.

I. INTRODUCTION

In many machine learning problems, the distance metric used over the input data has critical impact on the success of a learning algorithm. For instance, k-nearest neighbor (k-NN) classification [4], and clustering algorithms such as k-means rely on if an appropriate distance metric is used to faithfully model the underlying relationships between the input data points. A more concrete example is visual object recognition. Many visual recognition tasks can be viewed as inferring a distance metric that is able to measure the (dis)similarity of the input visual data, ideally being consistent with human perception. Typical examples include object categorization [25] and content-based image retrieval [20], in which a similarity metric is needed to discriminate different object classes or relevant and irrelevant images against a given query. As one of the most classic and simplest classifiers, k-NN has been applied to a wide range of vision tasks and it is the classifier that directly depends on a predefined distance metric. An appropriate distance metric is usually needed for achieving a promising accuracy. Previous works (e.g., [26] and [27]) have shown that compared to using the standard Euclidean distance, applying a well-designed distance often can significantly boost the classification accuracy of a k-NN classifier. In this work, we propose a scalable and fast algorithm to learn a Mahalanobis distance metric. Mahalanobis metric removes the main limitation of the Euclidean metric in that it corrects for correlation between the different features.

Recently, much research effort has been spent on learning a Mahalanobis distance metric from labeled data [5], [24], [26], [27]. Typically, a convex cost function is defined such that a global optimum can be achieved in polynomial time. It has been shown in the statistical learning theory [23] that increasing the margin between different classes helps to reduce the generalization error. Inspired by the work of [24], we directly learn the Mahalanobis matrix from a set of distance comparisons, and optimize it via margin maximization. The intuition is that such a learned Mahalanobis distance metric may achieve sufficient separation at the boundaries between different classes. More importantly, we address the scalability problem of learning the Mahalanobis distance matrix in the presence of high-dimensional feature vectors, which is a critical issue of distance metric learning. As indicated in a theorem in [19], a positive semidefinite trace-one matrix can always be decomposed as a convex combination of a set of rank-one matrices. This theorem has inspired us to develop a fast optimization algorithm that works in the style of gradient descent. At each iteration, it only needs to find the principal eigenvector of a matrix of size $D \times D$ ($D$ is the dimensionality of the input data) and a simple matrix update. This process incurs much less computational overhead than the metric learning algorithms in the literature [2], [24]. Moreover, thanks to the above theorem, this process automatically preserves the p.s.d. property of the Mahalanobis matrix. To verify its effectiveness and efficiency, the proposed algorithm is tested on a few benchmark data sets and is compared with the state-of-the-art distance metric learning algorithms. As experimentally demonstrated, k-NN with the Mahalanobis distance learned by our algorithms attains comparable (sometimes slightly better) classification accuracy. Meanwhile, in terms of the computation time, the proposed algorithm has much better scalability in terms of the dimensionality of input feature vectors.

We briefly review some related work before we present our method. Given a classification task, some previous work on learning a distance metric aims to find a metric that makes the data in the same class close and separates those in different classes from each other as far as possible. Xing et al. [26] proposed an approach to learn a Mahalanobis distance for supervised clustering. It minimizes the sum of the distances...
II. LARGE-MARGIN MAHALANOBSIS METRIC LEARNING

In this section, we propose our distance metric learning approach as follows. The intuition is to find a particular distance metric for which the margin of separation between the classes is maximized. In particular, we are interested in learning a quadratic Mahalanobis metric.

Let $a_i \in \mathbb{R}^d (i = 1, 2, \ldots, n)$ denote a training sample where $n$ is the number of training samples and $D$ is the number of features. To learn a Mahalanobis distance, we create a set $S$ that contains a group of training triplets as $S = (a_i, a_j, a_k)$, where $a_i$ and $a_j$ come from the same class and $a_k$ belongs to different classes. A Mahalanobis distance is defined as follows. Let $P \in \mathbb{R}^{d \times d}$ denote a linear transformation and $\text{dist}$ be the squared Euclidean distance in the transformed space. The squared distance between the projections of $a_i$ and $a_j$ writes

$$\text{dist}_{ij} = \|P^T a_i - P^T a_j\|^2 = (a_i - a_j)^T P^T P (a_i - a_j).$$ (1)

According to the class memberships of $a_i$, $a_j$, and $a_k$, we wish to achieve $\text{dist}_{ik} \geq \text{dist}_{ij}$ and it can be obtained as

$$(a_i - a_k)^T P^T (a_i - a_j) \geq (a_k - a_j)^T P^T (a_i - a_j).$$ (2)

It is not difficult to see that this inequality is generally not a convex constrain in $P$ because the difference of quadratic terms in $P$ is involved. In order to make this inequality constrain convex, a new variable $X = P^T$ is introduced and used throughout the whole learning process. Learning a Mahalanobis distance is essentially learning the Mahalanobis matrix $X$, (2) becomes linear in $X$. This is a typical technique to convexify a problem in convex optimization [2].

A. Maximization of Soft Margins

In our algorithm a margin is defined as the difference between $\text{dist}_{ik}$ and $\text{dist}_{ij}$, that is

$$\rho_r = (a_i - a_j)^T X (a_i - a_k) - (a_k - a_j)^T X (a_i - a_i) \quad \forall (a_i, a_j, a_k) \in S, \quad r = 1, 2, \ldots, |S|.$$ (3)

Similar to the large margin principle that has been widely used in machine learning algorithms such as support vector machines and boosting, here we maximize this margin (3) to obtain the optimal Mahalanobis matrix $X$. Clearly, the larger is the margin $\rho_r$, the better metric might be achieved. To enable some flexibility, i.e., to allow some inequalities of (2) not to be satisfied, a soft-margin criterion is needed. Considering these factors, we could define the objective function for learning $X$ as

$$\max_{\rho \geq 0, X} \rho - C \sum_{r=1}^{|S|} \xi_r, \quad \text{subject to}$$

$$X \succeq 0, \quad \text{Tr}(X) = 1,$$

$$\xi_r \geq 0, \quad r = 1, 2, \ldots, |S|,$$

$$(a_i - a_j)^T X (a_i - a_k) - (a_k - a_j)^T X (a_i - a_i) \geq \rho - \xi_r, \quad \forall (a_i, a_j, a_k) \in S.$$ (4)

The rest of this work is organized as follows. Section II presents the convex formulation of learning a Mahalanobis metric. In Section III, we show how to efficiently solve the optimization problem by a specialized gradient descent procedure, which is the main contribution of this work. The performance of our approach is experimentally demonstrated in Section IV. Finally, we conclude this work in Section V.
where $X \succeq 0$ constrains $X$ to be a p.s.d. matrix and $\text{Tr}(X)$ denotes the trace of $X$. $r$ indexes the training set $S$ and $|S|$ denotes the size of $S$. $C$ is an algorithmic parameter that balances the violation of (2) and the margin maximization. $\xi \geq 0$ is the slack variable similar to that used in support vector machines and it corresponds to the soft-margin hinge loss.

The above cost function is equivalent to the cost function in (4), which adopts the hinge loss function defined as

$$
\lambda \left( \langle A', X \rangle - \rho \right) =
\begin{cases}
0, & \text{if } \langle A', X \rangle - \rho \geq 0, \\
(\langle A', X \rangle - \rho)^2, & \text{if } \langle A', X \rangle - \rho < 0.
\end{cases}
$$

As shown in Fig. 1, this function connects the positive and zero segments smoothly and it is differentiable everywhere including the point $\rho = 0$. We also consider the Huber loss function in this work

$$
\lambda \left( \langle A', X \rangle - \rho \right) =
\begin{cases}
0, & \text{if } \langle A', X \rangle - \rho \geq h, \\
2 \left( \langle A', X \rangle - \rho \right)^2 - h^2, & \text{if } -h < \langle A', X \rangle - \rho < h, \\
-(\langle A', X \rangle - \rho)^2, & \text{if } \langle A', X \rangle - \rho \leq -h.
\end{cases}
$$

where $h$ is a parameter whose value is usually between 0.01 and 0.5. A Huber loss function with $h = 0.5$ is plotted in Fig. 1. There are three different parts in the Huber loss function, and they together form a continuous and differentiable function. This loss function approaches the hinge loss curve when $h \to 0$. Although the Huber loss is more complicated than the squared hinge loss, its function value increases linearly with the value of $\langle A', X \rangle - \rho$. Hence, when a training set containing outliers or samples heavily contaminated by noise, the Huber loss might give a more reasonable (milder) penalty than the squared hinge loss does. We discuss both loss functions in our experimental study. Again, we highlight that by using these two loss functions, the cost function $f(\tilde{X}, \rho)$ that we are going to optimization becomes differentiable with respect to both $X$.
Algorithm 1 The proposed optimization algorithm.

Input:
The maximum number of iterations $K$; 
A pre-set tolerance value $\varepsilon$ (e.g., $10^{-3}$).
1. Initialize $X_0$ such that $\text{Tr}(X_0) = 1$, $\text{rank}(X_0) = 1$;
2. for $k = 1, 2, \ldots, K$ do
   3. compute $\rho_k$ by solving the subproblem
   4. Compute $X_k$ by solving the problem
   5. if $k = 1$ and $f(X_k, \rho_k) = f(X_{k-1}, \rho_{k-1}) < \varepsilon$ and
      $f(X_k, \rho_k) - f(X_{k-1}, \rho_{k-1}) < \varepsilon$ then
   6. break (converged);
Output: The final p.s.d. matrix $X_k$.

Algorithm 2 Compute $X_k$ in the proposed algorithm.

Input: $\rho_k$ and $X_k$, which is an initial approximation of $X_k$; 
The maximum number of iterations $J$.
1. for $i = 1, 2, \ldots, J$ do
   2. compute $v_i$ that corresponds to the largest eigenvalue $l_i$
      of the matrix $\nabla f(X_k, \rho_k)$;
   3. if $l_i < \varepsilon$ then
   4. break (converged);
   5. let the search direction be $p_i = v_i^\top - X_k$;
   6. set $X_{k+1} = X_k + \alpha p_i$. Here, $\alpha$ is found by line search;
Output: Set $X_k = X_{k+1}$.

and $\rho$.

III. SCALABLE AND FAST OPTIMIZATION ALGORITHM

The proposed algorithm maximizes the objective function iteratively, and in each iteration the two variables $X$ and $\rho$ are optimized alternatively. Note that the optimization in this alternative strategy retains the global optimum because $f(X, \rho)$ is a convex function in both variables ($X, \rho$) and $(X, \rho)$ are not coupled together. We summarize the proposed algorithm in Algorithm 1. Note that $\rho_k$ is a scalar and line 3 in Algorithm 1 can be solved directly by a simple 1-D maximization process. However, $X$ is a p.s.d. matrix with size of $D \times D$. Recall that $D$ is the dimensionality of feature vectors. The following section presents how $X$ is efficiently optimized in our algorithm.

A. Optimizing for the Mahalanobis Matrix $X_k$

Let $P = \{X \in \mathbb{R}^{D \times D} : X \succeq 0, \text{Tr}(X) = 1\}$ be the domain in which a feasible $X$ lies. Note that $P$ is a convex set of $X$. As shown in line 4 in Algorithm 1, we need to solve the following maximization problem:

$$\max_{X \succeq 0} f(X, \rho_k)$$

(11)

where $\rho_k$ is the output of line 3 in Algorithm 1. Our algorithm offers a simple and efficient way for solving this problem by explicitly maintaining the positive semidefiniteness property of the matrix $X$. It needs only compute the largest eigenvalue and the corresponding eigenvector whereas most previous approaches such as the method of [24] require a full eigen-decomposition of $X$. Their computational complexities are $O(D^3)$ and $O(D^3)$, respectively. When $D$ is large, this computational complexity difference could be significant.

Let $\nabla f(X, \rho_k)$ be the gradient matrix of $f(\cdot)$ with respect to $X$ and $\alpha$ be the step size for updating $X$. Recall that we update $X$ in such a way that $X_{k+1} = (1 - \alpha)X_k + \alpha X$, where $\text{rank}(X_k) = 1$ and $\text{Tr}(X_k) = 1$. To find the $\Delta X$ that satisfies these constraints and in the meantime can best approximate the gradient matrix $\nabla f(X, \rho_k)$, we need to solve the following optimization problem:

$$\max_{X \in P} \langle \nabla f(X, \rho_k), \Delta X \rangle$$

subject to $\text{rank}(X) = 1$ and $\text{Tr}(X) = 1$. (12)

The optimal $\Delta X^*$ is exactly $v_1 v_1^\top$ where $v$ is the eigenvector of $\nabla f(X, \rho_k)$ that corresponds to the largest eigenvalue. The constraints say that $\Delta X$ is an outer product of a unit vector: $\Delta X = v v^\top$ with $\|v\|_2 = 1$. Here, $\|\cdot\|_2$ is the Euclidean norm. Problem (12) can then be written as: max $v_1^\top \nabla f(X, \rho_k)v$, subject to $\|v\|_2 = 1$. It is clear now that an eigen-decomposition gives the solution to the above problem.

Hence, to solve the above optimization, we only need to compute the leading eigenvector of the matrix $\nabla f(X, \rho_k)$. Note that $X$ still retains the properties of $X \succeq 0, \text{Tr}(X) = 1$ after applying this process.

Clearly, a key parameter of this optimization process is $\alpha$ which implicitly decides the total number of iterations. The computational overhead of our algorithm is proportional to the number of iterations. Hence, to achieve a fast optimization process, we need to ensure that in each iteration the $\alpha$ can lead to a sufficient reduction on the value of $f(\cdot)$. This is discussed in the following part.

B. Finding the Optimal Step Size $\alpha$

We employ the backtracking line search algorithm in [17] to identify a suitable $\alpha$. It reduces the value of $\alpha$ until the Wolfe conditions are satisfied. As shown in Algorithm 2, the search direction is $p_i = v_i^\top - X_k$. The Wolfe conditions that we use are

$$f(X_k + \alpha p_i, \rho_k) \leq f(X_k, \rho_k) + \alpha \nabla f(X_k, \rho_k)^\top p_i$$

$$\|p_i^\top \nabla f(X_k, \rho_k)\| \leq c_2 \|p_i\|^2$$

(13)

where $0 < c_1 < c_2 < 1$. The result of backtracking line search is an acceptable $\alpha$ which can give rise to sufficient reduction on the function value of $f(\cdot)$. We show in the experiments that with this setting our optimization algorithm can achieve better computational efficiency than some of the existing solvers.

IV. EXPERIMENTS

The goal of these experiments is to verify the efficiency of our algorithm in achieving comparable (or sometimes even better) classification performances with a reduced computational cost. We perform experiments on 10 data
sets described in Table I. For some data sets, principal component analysis (PCA) is performed to remove noises and reduce the dimensionality. The metric learning algorithms are then run on the data sets preprocessed by PCA. The Wine, Balance, Vehicle, Breast-Cancer and Diabetes data sets are obtained from University of California, Irvine, Machine Learning Repository [16]; USPS is from S. Roweis’ website;1 MNIST and Letter are from Libsvm [3]. For MNIST, we only use its test data in our experiment. The ORLface data is from AT&T research2 and Twin-Peaks is downloaded from L. van der Maaten’s website.3 The Face and Background classes (435 and 520 images, respectively) in the image retrieval experiment are obtained from the Caltech-101 object database [6]. In order to perform statistics analysis, the ORLface, Twin-Peaks, Wine, Balance, Vehicle, Diabetes, and Face-Background data sets are randomly split as 10 pairs of train/validation/test subsets and experiments on those data sets are repeated 10 times on each split.

The k-NN classifier with the Mahalanobis distance learned by our algorithm (termed SDPMetric in short) is compared with the k-NN classifiers using a simple Euclidean distance (“Euclidean” in short) and that learned by LMNN4 in [24]. Since, Weinberger et al. [24] have shown that LMNN obtains the classification performance comparable to support vector machines on some data sets, we focus on the comparison between our algorithm and LMNN, which is considered as the state-of-the-art. To prepare the training triplet set S, we apply the 3-NN method to these data sets and generate the training triplets for our algorithm. The training data sets for LMNN are also generated using 3-NN, except that the Twin-peaks and ORLface are applied with the 1-NN method. Also, the experiment compares the two variants of our proposed SDPMetric, which use the squared hinge loss (denoted as SDPMetric-S) and the Huber loss(SDPMetric-H), respectively. We split each data set into 70/15/15% randomly and refer to those split sets as training, cross validation and test sets except preseparated data sets (Letter) and Face-Background which was made for image retrieval. Following [24], LMNN uses 85/15% data for training and testing. The training data are also split into 70/15/15% in LMNN for cross validation to be consistent with our SDPMetric. The Letter data set is separated according to Hsu and Lin [9]. Same as in [24], PCA is applied to USPS, MNIST, and ORLface to reduce the dimensionality of feature vectors.

The following experimental study demonstrates that our algorithm achieves slightly better classification accuracy rates with a much less computational cost than LMNN on most of the tested data sets. The detailed test error rates and timing results are reported in Tables II and III. As we can see, the test error rates of SDPMetric-S are comparable to those of LMNN. SDPMetric-H achieves lower misclassification error rates than LMNN and the Euclidean distance on most of data sets except Face-Background data (which is treated as an image retrieval problem) and MNIST, on which SDPMetric-S achieves a lower error rate. Overall, we can conclude that the proposed SDPMetric either with squared hinge loss or Huber loss is at least comparable to (or sometimes slightly better than) the state-of-the-art LMNN method in terms of classification performance.

Before reporting the timing result on these benchmark data sets, we compared our algorithm (SDPMetric-H) with two convex optimization solvers, namely, SeDuMi [21] and SDPT3 [22] which are used as internal solvers in the disciplined convex programming software CVX [8]. Both SeDuMi and SDPT3 use interior-point based methods. To perform eigen-decomposition, our SDPMetric uses ARPACK [13], which is designed to solve large scale eigenvalue problems. Our SDP-Metric is implemented in standard C++. Experiments have been conducted on a standard desktop. We randomly generated 1000 training triplets and gradually increase the dimensionality of feature vectors from 20 to 100. Fig. 2 illustrates the computational time of ours, CVX/SeDuMi and CVX/SDPT3. As shown, the computational load of our algorithm almost keeps constant as the dimensionality increases. This might be because the proportion of eigen-decomposition’s CPU time does not dominate with dimensions varying from 20 to 100 in SDPMetric on this data set. In contrast, the computational loads of CVX/SeDuMi and CVX/SDPT3 increase quickly in this course. In the case of the dimension of 100, the difference on CPU time can be as large as 800 to 1000 s.

This shows the inefficiency and poor scalability of standard interior-point methods. Secondly, the computational time of LMNN, SDPMetric-S, and SDPMetric-H on these benchmark data sets are compared in Table III. As shown, LMNN is always slower than the proposed SDPMetric which converges very fast on these data sets. Especially, on the Letter and Twin-Peaks data sets, SDPMetric shows significantly improved computational efficiency. Note that for a fair comparison, same k-NN strategy is used to generate triplets for SDPMetric and pairwise constraints for LMNN in this experiments.

Face-Background data set consists of the two object classes, Face-easy and Background-Google in [6], as a retrieval prob-

---

1http://www.cs.nyu.edu/~roweis/data.html
2http://ticc.uvt.nl/lvdrmaaten/
3http://www.cs.nyu.edu/
4In our experiment, we have used the implementation of LMNN’s authors. Note that to be consistent with the setting in [24], LMNN here also uses the “t0 = 1” option and updates the projection matrix to speed up its computation. If we update the distance matrix directly to get global optimum, LMNN would be much slower due to full eigen-decomposition at each iteration.
TABLE I

Ten Benchmark Data Sets Used in the Experiment. Missing Entries in “Dimension After PCA” Indicate No PCA Processing

<table>
<thead>
<tr>
<th>Data Set</th>
<th># Training</th>
<th># Validation</th>
<th># Test</th>
<th>Dimension</th>
<th>Dimension After PCA</th>
<th># Classes</th>
<th># Runs</th>
<th># Triplets for SDPMetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPSPCA</td>
<td>7700</td>
<td>1650</td>
<td>1650</td>
<td>256</td>
<td>60</td>
<td>10</td>
<td>1</td>
<td>69500</td>
</tr>
<tr>
<td>USPS</td>
<td>7700</td>
<td>1650</td>
<td>1650</td>
<td>256</td>
<td>60</td>
<td>10</td>
<td>1</td>
<td>7750</td>
</tr>
<tr>
<td>MINSTPCA</td>
<td>7000</td>
<td>1500</td>
<td>1500</td>
<td>784</td>
<td>256</td>
<td>60</td>
<td>10</td>
<td>60000</td>
</tr>
<tr>
<td>MINST</td>
<td>7000</td>
<td>1500</td>
<td>1500</td>
<td>784</td>
<td>256</td>
<td>60</td>
<td>10</td>
<td>7000</td>
</tr>
<tr>
<td>Letter</td>
<td>10500</td>
<td>4500</td>
<td>5000</td>
<td>16</td>
<td>256</td>
<td>26</td>
<td>1</td>
<td>94500</td>
</tr>
<tr>
<td>ORLFace</td>
<td>280</td>
<td>60</td>
<td>60</td>
<td>256</td>
<td>42</td>
<td>40</td>
<td>10</td>
<td>260</td>
</tr>
<tr>
<td>Twin-Peaks</td>
<td>14000</td>
<td>3000</td>
<td>3000</td>
<td>3</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>14000</td>
</tr>
<tr>
<td>Wine</td>
<td>126</td>
<td>26</td>
<td>26</td>
<td>13</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>1134</td>
</tr>
<tr>
<td>Balance</td>
<td>419</td>
<td>93</td>
<td>93</td>
<td>4</td>
<td>3</td>
<td>10</td>
<td>1</td>
<td>3951</td>
</tr>
<tr>
<td>Vehicle</td>
<td>593</td>
<td>127</td>
<td>126</td>
<td>18</td>
<td>4</td>
<td>10</td>
<td>1</td>
<td>5337</td>
</tr>
<tr>
<td>Breast-Cancer</td>
<td>479</td>
<td>102</td>
<td>102</td>
<td>10</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>4311</td>
</tr>
<tr>
<td>Diabetes</td>
<td>538</td>
<td>115</td>
<td>115</td>
<td>8</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>4842</td>
</tr>
<tr>
<td>Face-Background</td>
<td>472</td>
<td>191</td>
<td>382</td>
<td>100</td>
<td>2</td>
<td>10</td>
<td>1</td>
<td>4428</td>
</tr>
</tbody>
</table>

Fig. 3. Retrieval performances of SDPMetric-S, SDPMetric-H, LMNN, and the Euclidean distance. The curves of SDPMetric-S and SDPMetric-H are very close.

TABLE II

3-Nearest Neighbor Misclassification Error Rates (%). The Standard Deviation Values Are in Brackets. The Best Results Are Highlighted in Bold

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Euclidean</th>
<th>LMNN</th>
<th>SDPMetric-S</th>
<th>SDPMetric-H</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPSPCA</td>
<td>2.85</td>
<td>2.70</td>
<td>2.70</td>
<td>1.70</td>
</tr>
<tr>
<td>USPS</td>
<td>4.42</td>
<td>3.50</td>
<td>3.50</td>
<td>2.31</td>
</tr>
<tr>
<td>MINSTPCA</td>
<td>5.15</td>
<td>3.15</td>
<td>3.00</td>
<td>3.35</td>
</tr>
<tr>
<td>MINST</td>
<td>4.80</td>
<td>4.80</td>
<td>4.13</td>
<td>4.53</td>
</tr>
<tr>
<td>Letter</td>
<td>5.38</td>
<td>4.04</td>
<td>3.60</td>
<td>5.60</td>
</tr>
<tr>
<td>ORLFace</td>
<td>6.00</td>
<td>5.00</td>
<td>4.75</td>
<td>4.34</td>
</tr>
<tr>
<td>Twin-Peaks</td>
<td>1.03</td>
<td>0.39</td>
<td>0.43</td>
<td>1.11</td>
</tr>
<tr>
<td>Wine</td>
<td>24.62</td>
<td>18.99</td>
<td>21.67</td>
<td>20.87</td>
</tr>
<tr>
<td>Balance</td>
<td>4.51</td>
<td>4.31</td>
<td>4.16</td>
<td>2.94</td>
</tr>
<tr>
<td>Vehicle</td>
<td>28.00</td>
<td>27.65</td>
<td>26.70</td>
<td>27.64</td>
</tr>
<tr>
<td>Breast-Cancer</td>
<td>28.00</td>
<td>27.65</td>
<td>26.70</td>
<td>27.64</td>
</tr>
<tr>
<td>Diabetes</td>
<td>28.00</td>
<td>27.65</td>
<td>26.70</td>
<td>27.64</td>
</tr>
<tr>
<td>Face-Background</td>
<td>28.00</td>
<td>27.65</td>
<td>26.70</td>
<td>27.64</td>
</tr>
</tbody>
</table>

TABLE III

Computational Time for Each Run

<table>
<thead>
<tr>
<th>Data Set</th>
<th>LMNN</th>
<th>SDPMetric-S</th>
<th>SDPMetric-H</th>
</tr>
</thead>
<tbody>
<tr>
<td>USPSPCA</td>
<td>256s</td>
<td>111s</td>
<td>256s</td>
</tr>
<tr>
<td>USPS</td>
<td>16h</td>
<td>106h</td>
<td>20n</td>
</tr>
<tr>
<td>MINSTPCA</td>
<td>210s</td>
<td>111s</td>
<td>99s</td>
</tr>
<tr>
<td>MINST</td>
<td>9.7h</td>
<td>14s</td>
<td>57n</td>
</tr>
<tr>
<td>Letter</td>
<td>108s</td>
<td>6s</td>
<td>156s</td>
</tr>
<tr>
<td>ORLFace</td>
<td>13s</td>
<td>less than 1s</td>
<td>less than 1s</td>
</tr>
<tr>
<td>Twin-Peaks</td>
<td>95%</td>
<td>less than 1s</td>
<td>less than 1s</td>
</tr>
<tr>
<td>Wine</td>
<td>9s</td>
<td>2s</td>
<td>2s</td>
</tr>
<tr>
<td>Balance</td>
<td>7s</td>
<td>less than 1s</td>
<td>2s</td>
</tr>
<tr>
<td>Vehicle</td>
<td>19s</td>
<td>2s</td>
<td>7s</td>
</tr>
<tr>
<td>Breast-Cancer</td>
<td>4s</td>
<td>2s</td>
<td>3s</td>
</tr>
<tr>
<td>Diabetes</td>
<td>6s</td>
<td>less than 1s</td>
<td>2s</td>
</tr>
<tr>
<td>Face-Background</td>
<td>92s</td>
<td>5s</td>
<td>9s</td>
</tr>
</tbody>
</table>
The results are reported in Fig. 4. Second, we keep the number of triplets fixed to 7700 and vary the feature dimension after PCA. For the first setting, as expected, when more triplets are used, the test error decreases from 2.85% to 1.70%. The computational time increases with more triplets, from 5 s to 491 s. For the second setting, we see that with the dimension being 60, 120, and 256, the computational time for each case is 5, 63, and 120 s, respectively. The error rates in this setting show that PCA does help improve the classification performance on USPS, if the dimension is properly chosen, probably due to noise reduction. The PCA dimension might need to be tuned by cross validation.

V. CONCLUSION

We have proposed a new algorithm to demonstrate how to efficiently learn a Mahalanobis distance metric with the principle of margin maximization. Enlightened by the important theorem on p.s.d. matrix decomposition in [19], we have designed a gradient descent method to update the Mahalanobis distance metric with the p.s.d. property of the learned matrix is maintained during the whole optimization process. Experiments on benchmark data sets and the retrieval problem verify the superior classification performance and computational efficiency of the proposed distance metric learning algorithm.

The proposed algorithm may be used to solve more general SDP problems in machine learning. To look for other applications is one of the future research directions.

REFERENCES