Distributed Information-Theoretic Metric Learning in Apache Spark

Yuxin Su, Haiqin Yang, Irwin King and Michael Lyu
Shenzhen Key Laboratory of Rich Media Big Data Analytics and Applications,
Shenzhen Research Institute, The Chinese University of Hong Kong, Shenzhen, China
Department of Computer Science and Engineering
The Chinese University of Hong Kong, Shatin, Hong Kong
Email: {yxsu, hqyang, king, lyu}@cse.cuhk.edu.hk

***Abstract*—Distance metric learning (DML) is an effective similarity learning tool to learn a distance function from examples to enhance the model performance in applications of classification, regression, and ranking, etc. Most DML algorithms need to learn a Mahalanobis matrix, a positive semidefinite matrix that scales quadratically with the number of dimensions of input data. This brings huge computational cost in the learning procedure, and makes all proposed algorithms infeasible for extremely high-dimensional data even with the low-rank approximation. Differently, in this paper, we take advantage of the power of parallel computation and propose a novel distributed distance metric learning algorithm based on a state-of-the-art DML algorithm, Information-Theoretic Metric Learning (ITML). More specifically, we utilize the property that each positive semidefinite matrix can be decomposed into a combination of rank-one and trace-one matrices and convert the original sequential training procedure into a parallel one. In most cases, the communication demands of the proposed method are also reduced from $\mathcal{O}(d^2)$ to $\mathcal{O}(cd)$, where d is the number of dimensions of the data and c is the number of constraints in DML and can be smaller than d by appropriate selection. Moreover importantly, we present a rigorous theoretical analysis to upper bound the Bregman divergence between the sequential algorithm and the parallel algorithm, which guarantees the correctness and performance of the proposed algorithm. Our experiments on datasets with $\mathcal{O}(10^5)$ features demonstrate the competitive scalability and the performance compared with the original ITML algorithm.**

Introduction

Learning an appropriate distance metric is an important topic in both machine learning and data mining. Distance metric learning has been widely applied in many problems, such as image retrieval [1], face recognition [1], bioinformatics analysis [2], software error detection [3], [4]. Earlier publications formulate the learning problem as an convex optimization problem by maximizing the sum of difference between the dissimilar instances while restricting the distance between the similar instances to be small [5]. However, solving such problem by semidefinite programming (SDP) solver or eigenvalue decomposition is time-consuming and poorly scalable to medium and high dimensional data. Recent developments such as Large Margin Nearest Neighbors (LMNN) [6], [7], BoostMetric [8] and Information-Theoretic Metric Learning (ITML) [3] try to speedup DML by the special structure of Mahalanobis matrix, partial eigenvalue decomposition or low-rank approximation.

However, Most of existing algorithms are incapable of handling high-dimensional data. LMNN is very easy to overfit for high-dimensional data. BoostMetric employing the max eigenvalue decomposition needs a huge number of iterations to converge for high-dimensional data. ITML conducts heavy matrix multiplications which is very time-consuming for high-dimensional data.

For other earlier work, is a popular solution for DML problem but it is very easy to overfit especially for high-dimensional data. [9] relies on the special structure of Mahalanobis matrix. [8] is an efficient DML solution only related with the max eigenvalue decomposition, however, it needs a huge number of iterations to converge for high-dimensional data.

In the high dimensional setting, most existing work mainly exploit the low-rank structure of the learned matrix or the sparsity of the covariance matrix. Instead of learning the full rank matrix $\mathbf{A}$, [10] factorizes the matrix $\mathbf{A}$ as $\mathbf{R}\mathbf{R}^T$ and learned a low rank approximation of the Mahalanobis matrix. [11] imposes a different prior on learning the Mahalanobis distance based on the sparsity of sample concentration matrix Σ^{-1}, where Σ is the covariance matrix of samples. [12] proposed the low dimensional projection to extract a compact feature representation for originally high dimensional data. However, all the above properties may not hold for highly complex data in real-life applications. Although each of these algorithm was shown to yield excellent performance in classification and clustering tasks, they do not generalize learning an arbitrary metric, parameterized by an arbitrary matrix. To the best of our knowledge, it is still nontrivial to learn the approximated distance between two instances with high dimensionality if the low-rank property is not satisfied. Hence, how to learn the metric without highly depending on the low-rank assumption still requires sophisticated designs and experimental explorations.

Meanwhile, with large volumes of data available in many real life applications, the problem of scalability becomes the key concern in designing many machine learning algorithms. Most of the existing work in scalable machine learning relies on approximation methods and parallel computing schemes. Specifically, in term of the algorithm aspect, most existing work mainly focus on stochastic gradient descent method for convex optimization problem [13], [14], [15] or proximal descent method for non-convex problem [16], [17], [18]. Few effort has been done on solving the semi-definite programming problem for matrices in a distributed cluster, especially for DML problem. Very recently, [19] proposed a distributed approach to speedup the computation of DML problem. Unfortunately, both of its methodology and proofs heavily rely on the low-rank property of Mahalanobis matrix. Forcing low-rank assumption without understanding the data may destroy the completeness of the data and yield suboptimal performance.

To address the above two challenges, taking advantage of powerful multi-thread cluster, we will explore to design a general parallel approach to solve DML problem for high-dimensional data without low-rank assumption about the Mahalanobis matrix. In this paper, we will design a scalable solution to speedup the training of ITML, which repeats Bregman projection [20]. By decomposing the Mahalanobis matrix into a linear combination of rank-one matrices, and dispatching them into different machines like a divide-and-conquer scheme, our implementation performs Bregman projection on these rank-one matrices without low-rank assumption about the Mahalanobis matrix. Finally, after iterations, the Mahalanobis matrix can be reconstructed from these rank-one matrices to calculate the pair-wise distances.

In summary, the proposed work contributes on the following aspects:

- By making a trade-off between accuracy and running time, we first propose a scalable approach to learn metrics from high-dimensional data without low-rank assumption.
- We compare the proposed method with the original ITML method from a theoretical perspective and give an upper error bound brought by our parallel update.
- The experiments conducted on the data with different scales demonstrate the correctness and the scalability of the our method.

Related Work

Since Xing et al, [5] formulate the distance metric learning problem as an optimization problem and solve it by semi-definite programming, there are many DML algorithms in the literature, which are developed for different objectives, such as learning global metric [8], local metric [21] or any special structured metric [22]. Completed surveys from different perspectives can be found in [4], [23]. Some algorithms focus on pairs-wise constraints [24], [25], some solves the optimization problem with relative constraints [6]. Due to the high complexity of computation, the majority of DML algorithms target low-dimensional or intermediate dimensional data up to hundreds of dimensions.

Several algorithms [11], [26] attempt to meet the challenge of high-dimensional metric learning. [27] assumes the Mahalanobis matrix as low-rank matrix and represented as $\mathbf{R}\mathbf{R}^T$. Meanwhile, [11] takes another form of low-rank assumption that the Mahalanobis matrix is sparse. [28] reduces the computational complexity during the eigendecomposition by picking up the largest eigenvalue and the corresponding eigenvector in a variant Frank-Wolfe algorithm [29]. [30] targets to the similar objective by taking the top-k eigenvalues and those corresponding eigenvectors. [26] simplifies the high-dimensional metric learning with multiple stages. [31] reformulates the high-dimensional Mahalanobis matrix as a combination of low-rank matrices, which are computed from gradient boosting as weak learners. In this paper, we also employ the similar formulation but with different approach to update the low-rank matrices.

Formulation

The distance metric learning task could be formulated as learning a Mahalanobis distance from a set of similar and dissimilar pairs of sample data. The Mahalanobis distance is defined as follows:

$$d(x,y) = \sqrt{(x-y)^T A (x-y)} \quad \forall x, y \in \mathbb{R}^n \tag{1}$$

where $A \in \mathcal{S}_+$. In most cases, we only take care about the squared Mahalanobis distance, which is denoted as d_A in the following context. Now the problem is translated to learn the positive semi-definite matrix A through the input data point. The input data are two sets representing pairs of similar data points and pairs of dissimilar data points respectively.

Given n instances $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ in $\mathbb{R}^d$, we aims at learning the matrix A which is involved in the Mahalanobis distance function $d_A(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i - \mathbf{x}_j)^T A (\mathbf{x}_i - \mathbf{x}_j)$.

Suppose that we have the similar and dissimilar pairs of instances $(\mathbf{x}_i, \mathbf{x}_j)$ in the set $\mathcal{S}$ and $\mathcal{D}$ respectively. We would like to keep the distance of the similar pairs small, i.e. $d_A(\mathbf{x}_i, \mathbf{x}_j) \le u$ when $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{S}$, while separate the dissimilar pairs as possible, i.e. $d_A(\mathbf{x}_i, \mathbf{x}_j) \ge l$ when $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D}$, where u and l are constant.

$$\mathcal{S} = \{(x_i, x_j) \ : \ d_A(x_i, x_j) \le u\}$$

$$\mathcal{D} = \{(x_i, x_j) \ : \ d_A(x_i, x_j) \ge l\}$$

Information-Theoretic Metric Learning (ITML)

ITML is one of the start-of-art algorithm for distance metric learning. It brings an assumption that the data come from multivariate Gaussian distribution, which is very common in most applications. Under this assumption, the Mahalanobis distance matrix A can be expressed as the inverse of the covariance matrix of the multivariate Gaussian distribution. Because the probability density function of multivariate Gaussian can be reformulated as $p(x; A) = \frac{1}{Z}\exp(-\frac{1}{2} d_A(x, \mu))$. Hence, the original DML problem is expressed as finding a proper multivariate Gaussian by minimizing difference relative entropy with the initial Mahalanobis distance function. The objective function is reformulated as:

$$\begin{array}{ll} \min & KL(p(x;A_0)||p(x;A) \\ s.t. & d_A(x_i,x_j) \le u \quad (i,j) \in S \\ & d_A(x_i,x_j) \ge l \quad (i,j) \in D \end{array}$$

where,

$$\begin{aligned} & KL(p(x|\mu_0,A_0)||p(x|\mu,A) \\ = & \int p(x|\mu_0,A_0) \log \frac{p(x|\mu_0,A_0)}{p(x|\mu,A)} dx \end{aligned} \tag{2}$$

A_0 is usually assigned to the identity matrix.

More specifically, the KL divergence in Eq.(2) can be expressed as a convex combination between a Mahalanobis distance between means and the Bregman divergence with respect to function $\phi(A) = -\log\det(A)$ between covariance matrices [32]:

$$KL(p(x|\mu_0,A_0)||p(x|\mu,A) = \frac{1}{2}D_\phi(A,A_0) + \frac{1}{2}d_{\Sigma^{-1}}(\mu_0,\mu) \tag{3}$$

where the Bregman divergence is defined as:

$$D_\phi(A,A_0) = \phi(A) - \phi(A_0) - \text{tr}\left(\nabla\phi(A_0)^T (A - A_0)\right) \tag{4}$$

With the assumption that the mean vectors of two Gaussian distribution are the same, ITML formulates the DML problem as:

$$\begin{array}{lll} \min & D_\phi(\mathbf{A},\mathbf{A}_0) & \\ s.t. & \text{tr}(\mathbf{A}(\mathbf{x}_i-\mathbf{x}_j)(\mathbf{x}_i-\mathbf{x}_j)^T) \le u, & (\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{S} \\ & \text{tr}(\mathbf{A}(\mathbf{x}_i-\mathbf{x}_j)(\mathbf{x}_i-\mathbf{x}_j)^T) \ge l, & (\mathbf{x}_i,\mathbf{x}_j) \in \mathcal{D} \end{array} \tag{5}$$

This problem is then cast into a particular Bregman Matrix Divergence problem [33], which contains global optimal solution and can be solved efficiently by Eq.(6) without eigenvalue computations or semi-definite programming.

$$A_{t+1} = A_t + \beta A_t(\mathbf{x}_i-\mathbf{x}_j)(\mathbf{x}_i-\mathbf{x}_j)^T A_t \tag{6}$$

where β is a scalar for the learning rate. The actual distance between a pair of data sample (x_i,x_j) in each iteration is computed from:

$$p(t) = (x_i-x_j)^T A_t (x_i-x_j) \tag{7}$$

Parallel Distance Metric Learning

The algorithm (see details in [3]) repeats the Bregman projection in Eq.(6) with different constraints sequentially. For each projection, it involves the computation on $A_t \in \mathbb{R}^{d\times d}$, where d is the number of dimensions of the input data. When d is large enough, the running time of the Bregman projection in single PC is not acceptable. That why the previous work with sequential update is only capable of handling the data with $\mathcal{O}(10^2)$-dimension.

In this paper, we try to conduct the Bregman projection with the power of parallel computation. The key step in Eq.(6) is to update the Mahalanobis matrix which is a $d \times d$ positive semi-definite matrix. Typically, there are two roads to parallel this update: 1) compute the matrix multiplication in Eq.(6) directly in distributed way. For example, the matrix is decomposed into blocks which are dispatched into different machines. When the local computation with respect to blocks finishes, the intermediate results are collected to form a complete matrix update. The drawback of this kind of approaches is obvious that there are heavy communications demand for carrying out the data in blocks; 2) design a Hogwild! [34]-like approach to update the Mahalanobis matrix based on partial information simultaneously. Unfortunately, the positive semi-definite property of Mahalanobis matrix is difficult to hold from multiple partial updates.

In order to hold the positive semi-definite property, we borrow the idea from [8] that any positive semi-definite matrix can be decomposed into a linear combination of rank-one matrices. Therefore, we have proposed a novel parallel solution for DML problem by reformulating the Mahalanobis matrix A as $A = I + \sum_i \alpha_i z_i z_i^T$ where $z_i \in \mathbb{R}^d$. By observing the update in Eq.(6), it is easy to find that $A_t(x_i - x_j)$ is a vector with d dimension and $A_t(x_i-x_j)(x_i-x_j)^T A^T$ is a rank-one matrix. Hence, we can concrete the formula of A by summing over the Bregman projection through all pairs of constraints in Eq.(8)

$$A_{t+1} = I + \sum_{i=1}^{C} \beta_i(t) z_i(t) z_i^T(t) \tag{8}$$

where, I is the unit matrix with the size of $d \times d$, C is the number of pairs of constraints, β is the step from Bregman projection in Eq.(6) and $z_i(t) = A_t c_i$. $c_i = x_j - x_k$ for the i-th pair of constraint (x_j, x_k). Actually, in the proposed framework, we only store z rather than A_t because A_t is a huge matrix when d becomes large. Consequently, the update of z is changed to Eq.(9)

$$\begin{aligned} z_k(t+1) &= A_{t+1} c_k \\ &= \left(I + \sum_{i=1}^{C} \beta_i(t) z_i(t) z_i^T(t) \right) c_k \end{aligned} \tag{9}$$

Based on the original algorithm in [3], $\beta_i(t)$ is a function of $p_i(t)$ in Eq. (7) and the upper bound or lower bound of constraints (see the details in Algorithm 1). The key step of updating $\beta_k(t)$ is to compute the actual distance $p_k(t)$. Combined with Eq.(8), the actual distance for the k-th constraint c_k can be expressed in Eq.(10)

$$\begin{aligned} p_k(t) &= c_k^T A_t c_k \\ &= c_k^T \left(I + \sum_{i=1}^{C} \beta_i(t) z_i(t) z_i^T(t) \right) c_k \\ &= c_k^T c_k + \sum_{i=1}^{C} \beta_i(t) c_k^T z_i(t) z_i^T(t) c_k \end{aligned} \tag{10}$$

Due to the separable property of the Mahalanobis matrix A in Eq. (8), we can dispatch the tasks for updating z in Eq.(9)

Algorithm 1 Parallel Distance Metric Learning

```
Input: $S$ : set of similar pairs; $D$: set of dissimilar pairs; $u, l$ :
    distance thresholds; $\gamma$ : slack parameter
Output: $A$ : Mahalanobis matrix
 1: $A = I$, $C = |S| + |D|$
 2: for constraint $(x_p, x_q)_k$,   $k \in \{1, 2, \dots, C\}$ do
 3:     $\lambda_k \leftarrow 0$
 4:     $d_k \leftarrow u$ for $(x_p, x_q)_k \in S$ otherwise $d_k \leftarrow l$
 5:     $c_k \leftarrow (x_p - x_q)_k$, $z_k \leftarrow c_k$
 6: end for
 7: while $\beta$ does not converge do
 8:     for all worker $k \in \{1, 2, \dots, C\}$ do in parallel
 9:         $z_k = c_k + \sum_{i=1}^{C} \beta_i z_i z_i^T c_k$
10:         $p \leftarrow c_k^T z_k$
11:         if $(x_p, x_q)_k \in S$ then
12:             $\alpha \leftarrow \min\left(\lambda_k, \frac{1}{2}\left(\frac{1}{p} - \frac{\gamma}{d_k}\right)\right)$
13:             $\beta \leftarrow \frac{\alpha}{1-\alpha p}$
14:             $d_k \leftarrow \frac{\gamma d_k}{\gamma + \alpha d_k}$
15:         else
16:             $\alpha \leftarrow \min\left(\lambda_k, \frac{1}{2}\left(\frac{\gamma}{d_k} - \frac{1}{p}\right)\right)$
17:             $\beta \leftarrow \frac{-\alpha}{1+\alpha p}$
18:             $d_k \leftarrow \frac{\gamma d_k}{\gamma - \alpha d_k}$
19:         end if
20:         $\lambda_k \leftarrow \lambda_k - \alpha$
21:         $z_k \leftarrow \left(I + \sum_{i=1}^{C} \beta_i z_i z_i^T\right) c_k$
22:         send $z_k$ to other workers.
23:     end for
24: end while
25: $A = I + \sum_{i=1}^{C} \beta_i z_i z_i^T$
```

and p in Eq.(10) into C workers which is an abstract concept representing the parallel executors, such as process in a single machine or different machines.

In our framework, worker k needs to receive all the update of z from other workers at the previous iteration first, then computes the update for the next iteration.

Overall, each work only sends a vector z rather than the matrix $\beta A(x_i - x_j)(x_i - x_j)^T A$ and receive $(C-1)$ times z rather than the whole Mahalanobis matrix in the original sequential algorithm. Hence, the communications demand for each work reduces from $\mathcal{O}(d^2)$ to $\mathcal{O}(cd)$. When the number of dimensions exceeds the number of constraints, which is a very commonly happened situation for high-dimensional metric learning tasks, the demand of communications reduces significantly compared with the original ITML algorithm.

Furthermore, when the number of constraints keep relative small, our work is equivalent to a ITML-based method with low-rank assumption about the Mahalanobis matrix. If the number of constraints becomes large, even larger than the number of dimensions, our method still works although the cost for communications will increase inevitably. Therefore, our method is a general parallel approach no matter the Mahalanobis matrix is low-rank or not.

IMPLEMENTATION

We implement our method on Apache Spark [35], which is a general propose, high-efficient, distributed in-memory computing platform. It provides a flexible data structure called Resilient Distributed Dataset (RDD) to support parallel computing on RDDs. The data or parameters in RDDs are cached in memory instead of loading from disk at each iteration like Hadoop [1]. Usually, the content of RDD is distributed in different machines and will be re-dispatched automatically without manual operation. Therefore, Spark is suitable for developers who only need to provide high-level logic to parallel the algorithm instead of caring about many details in cluster.

In our implementation, the list of z and the corresponding β are stored in a RDD. For each element of this RDD, we assign a "mapper" function to compute the update of z in Eq.(9) and the partial distance in Eq.(10) . All these "mapper" function will generate a new RDD to store the new z for next iteration. We also declare a constant "Broadcast" variable which is shared for all workers to store constraints c, because the computation of new z in each worker needs its corresponding c and the computation of partial distance iterates all constraints. In most cases, the "Broadcast" variable is cached in memory and can be fetched efficiently. Finally, another special component in Spark, called "Accumulator" will be created to collect the partial distance from each workers.

With the help of the global "Broadcast" and "Accumulator", the whole distributed algorithm is conducted in bulk synchronous parallel (BSP) [36], which guarantees that all parameters will be updated within the same iteration. More specifically, the employment of "Broadcast" and "Accumulator" in our implementation illustrated in Fig. 1 significantly reduce the redundant computation of the distance between each pairs of data sample under the learned metric space. Because the procedure of computing distance is divided into blocks which are shared among all workers.

THEORETICAL ANALYSIS

In this section, we will analyze the difference and give the upper bound of the proposed algorithm compared with the sequential algorithm in [3]. Because the result of the parallel computation of z in our algorithm may be different with the sequential version of the original algorithm.

First, we will compare the difference between the sequential version and the parallel version. In sequential order, the algorithm conducts C times Bregman projection to cover all pairs of constraints. After C iterations since iteration t, the Mahalanobis matrix A_{t+C} can be expressed as:

$$A_{t+C} = A_t + \sum_{i=1}^{C} \beta_i A_{t+i-1} c_i c_i^T A_{t+i-1}^T \tag{11}$$

For parallel version algorithm, if we rearrange the order of computation like a series of sequential updates, one iteration in

[1]https://hadoop.apache.org/

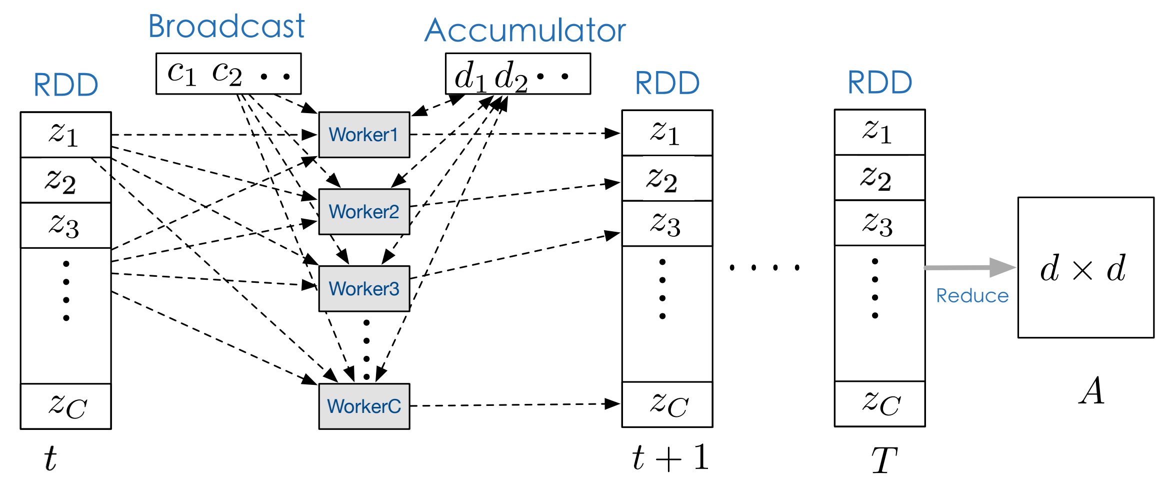

Figure 1. Framework of Parallel Distance Metric Learning on Apache Spark

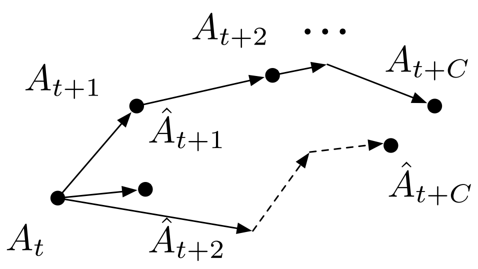

Figure 2. Delayed vs Normal Versions of Projection

parallel version algorithm is equal to C iterations in sequential algorithm but with the same A_t rather than A_{t+i-1}. The formula of parallel algorithm is expressed in Eq.()

$$\hat{A}_{t+C} = A_t + \sum_{i=1}^{C} \beta_i A_t c_i c_i^T A_t^T \tag{12}$$

By observing the difference between Eq.(11) and Eq.(12), we can find that the Bregman projection related with the i-th constraint in parallel algorithm is applied on a i-delayed Mahalanobis matrix. Figure 2 illustrates the difference between the normal update and the delayed update.

For convenience, we use the sequential version with delayed update to represent the parallel algorithm.

Theorem 1. *Delayed Update under Bregman Divergence*

Assume the difference of matrices is measured by the Bregman divergence with respect to LogDet divergence $\phi(X) = -\log\det(X)$. The minimizer of $D_\phi(A_0, A)$ after T iterations is A^*. The length of convergence path is denoted as Ω. The upper error bound of Algorithm 1 is

$$R[A] := \sum_{t=1}^{T} D_\phi(A_t, A^*) \le \frac{1}{\beta_{\min}} D_\phi(A^*, I) + \frac{1}{2} L\Omega$$

Proof. From the definition of Bregman divergence and the convexity of function $\phi(A)$ we know:

$$\begin{aligned} R[A] &= \sum_{t=1}^{T} D_\phi(A_t, A^*) \\ &= \sum_{t=1}^{T} \phi(A_t) - \phi(A^*) - \mathrm{tr}(\nabla\phi(A^*)^T (A_t - A^*)) \\ &\le \sum_{t=1}^{T} \phi(A_t) - \phi(A^*) \\ &\le \sum_{t=1}^{T} \mathrm{tr}\left(\nabla\phi(A_t)^T (A_t - A^*)\right) \\ &= \sum_{t=\tau}^{T+\tau} \mathrm{tr}\left(\lambda_{t-\tau}^T (A_t - A^*)\right) \end{aligned}$$

From Lemma 10 in [37], we have easily derived the matrix version as:

$$\begin{aligned} \mathrm{tr}\left(\lambda_{t-\tau}^T (A_t - A^*)\right) &\le \frac{1}{\beta_{t-\tau}} (D_\phi(A^*, A_t) - D_\phi(A^*, A_{t+1})) \\ &\quad + \beta_{t-\tau} \frac{||\lambda_{t-\tau}||_\star^2}{2} \\ &\le \frac{1}{\beta_{\min}} (D_\phi(A^*, A_t) - D_\phi(A^*, A_{t+1})) \\ &\quad + \beta_{t-\tau} \frac{||\lambda_{t-\tau}||_\star^2}{2} \end{aligned} \tag{13}$$

where $|| \cdot ||_\star$ is the Fenchel-Legendre dual norm [38].

Sum up these inequality in Eq.(13) as :

$$\begin{aligned}
&\sum_{t=\tau}^{T+\tau} \mathrm{tr}\left(\lambda_{t-\tau}^T A - A^*\right) \\
\leq& \frac{1}{\beta_{\min}} \sum_{t=\tau}^{T+\tau} \left(D_\phi(A^*, A_t) - D_\phi(A^*, A_{t+1})\right) \\
&+ \sum_{t=\tau}^{T+\tau} \beta_{t-\tau} \frac{||\lambda_{t-\tau}||_\star^2}{2} \\
=& \frac{1}{\beta_{\min}} \left(D_\phi(A^*, A_t) - D_\phi(A^*, A_T)\right) + \sum_{t=\tau}^{T+\tau} \beta_{t-\tau} \frac{||\lambda_{t-\tau}||_\star^2}{2} \\
\leq& \frac{1}{\beta_{\min}} D_\phi(A^*, I) + \frac{1}{2} \sum_{t=0}^{T} \beta_t ||\lambda_t||_\star^2 \\
\leq& \frac{1}{\beta_{\min}} D_\phi(A^*, I) + \frac{L}{2} \sum_{t=0}^{T} \beta_t ||\lambda_t||_\star \\
\leq& \frac{1}{\beta_{\min}} D_\phi(A^*, I) + \frac{1}{2} L\Omega
\end{aligned}$$

When $t \leq \tau$, $A_t = A_0 = I$ and $L = \sum_{t=0}^{T} \beta_t ||\lambda_t||_\star$ is the length of the path of the Bregman projection in the original algorithm without any delayed update. Typically, L has a upper bound Ω as long as the original algorithm converges in the convex set. $\square$

EXPERIMENTS

In this section, we first demonstrate the scalability of our proposed DITML algorithm on Apache Spark by comparing it with 1) the original ITML algorithm and 2) its implemented in a parallel scheme on high-dimensional data from four synthetic datasets and the ImageNet dataset. The k-NN classifiers ($k = 1$ and $k = 5$) are employed to evaluate the accuracy of the learned distance metric space.

Experimental Settings

All implementations are written in Scala 2.10 and run on Apache Spark 1.6.0 with YARN as the cluster controller on 32 physical machines with 4TB memory and 668 executors in total. All 32 physical machines are inter-connected with 10Gbps network switch. To make a fair comparison, we also implement the original ITML with the help of distributed matrix Scala classes in Spark 1.6.0.

Synthetic Datasets

We generate four sets of synthetic datasets on binary classification with dimensions in $10^{[2:1:5]}$ to test the scalability of our proposed framework with respect to the number of dimensions. More specifically, we employ the Scala object “KMeansDataGenerator” of “MLLib” package in Apache Spark to evenly generate the synthetic data following the normal distribution. The number of generated data for each dataset is one-tenth of the corresponding dimension. That is, the sizes of the datasets are $10, 100, 1000, 10000$, respectively. For each dataset, we randomly sample pairs of data points from the same class to generate similarity constraints, where the upper bound u is assigned to the Euclidean distance between the pairs. The dissimilarity constraints are sampled from pairs of data points belonging to different classes, where the lower bound l is assigned to the Euclidean distance between the dissimilar pairs.

ImageNet Dataset

For the ImageNet dataset, we randomly choose 50 pictures from two randomly selected classes, where each of them has 25 pictures as training data. DeCAF features [39], which capture most of the images information, are employed to represent the images in the experiment. The DeCAF features are extracted from a deep convolutional network. We then obtain 51,456 DeCAF features for each image in the test.

To determine the upper bound u and the lower bound l, we use the median value of Euclidean distance as a threshold for all pairs of data following the standard setup in [23]. For the pair of data belong to the same class, if the Euclidean distance is greater than the median value, then we assign the Euclidean distance as the upper bound of the similar pair, else assign the median value as the upper bound of the similar pair. For the pair of data belong to different classes, the rule is similar to that for the similar pairs. If the Euclidean distance is greater than the median value, then we assign the median value as the lower bound of the dissimilar pair, else we assign the Euclidean distance as the lower bound of the dissimilar pair. Overall, we have generated 1225 pair of constraints to train the Mahalanobis matrix in DML problem.

Scalability of Our Framework

Figure 4 shows the running time of our method with respect to the parallel execution on the Spark cluster. We have the following observations: 1) When the number of workers is less than 400, the total running time decreases significantly when the number of workers increases. 2) When the number of workers exceeds 400, the communication dominates the whole process and the speedup is not obvious when new workers come.

In order to display the advantage of the proposed DITML algorithm on the scalability issue compared with the original ITML, we implement the original ITML in Spark and transform the sequential execution of ITML to a distributed way by involving a distributed matrix class called “BlockMatrix”, which is a primitive data type in “MLLib” package in Apache Spark. Figure 3 displays direct comparisons of two kinds of ITML and the advantage of the proposed DITML algorithm compared with the original ITML on the execution time is significant. Overall fig. 3 and fig. 4 illustrate a good scalability of our method.

Performance of The Learned Metric Space

For the performance issue, Table I shows the accuracy of k-NN classification on the 4 synthetic datasets and the ImageNet. We use the Mahalanobis matrix learnt from our method to measure the distance between two samples, which is used in k-NN classification. Compared with the standard k-NN method

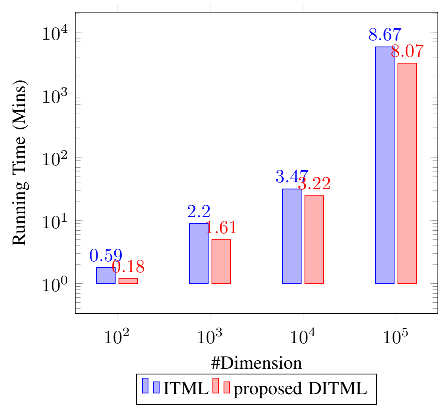

Figure 3. Comparisons between the original ITML and proposed DITML algorithms on different scaled synthetic datasets. Both of two algorithms iterates 50 times over all constraints.

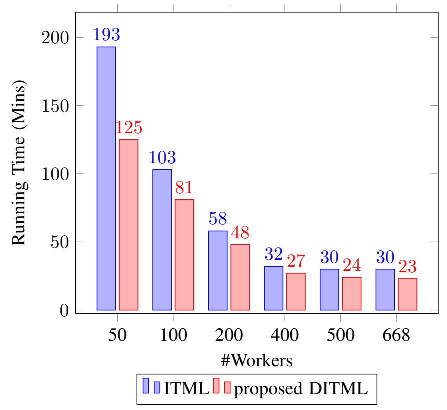

Figure 4. Comparisons between the original ITML and proposed DITML algorithms on 10000-dimensional synthetic data with different number of workers. Both of two algorithms iterates 50 times over all constraints.

without changing the metric, the proposed method enhances the accuracy of k-NN classification from 0.682 to 0.83 on ImageNet dataset.

The comparison between ITML and the proposed method displayed in Fig. 5 shows both of the algorithms will converge with the similar number of iterations. In these experiments, the slack variable γ is assigned to 100. Table I also demonstrates the proposed method is able to improve the performance of k-NN classification and the accuracy gap between these two algorithms is relative small.

Conclusion

In this paper, based on the observation that the positive semi-definite Mahalanobis matrix can be decomposed into a series of rank-one matrices, we have developed a parallel distance metric learning algorithm called DITML in order to learn the proper metric from high-dimensional data without low-rank approximation, which is infeasible for previous DML algorithms. Furthermore, the performance gap between the proposed method and the original ITML method is bounded from the theoretical perspective and can be ignored in actual experiments.

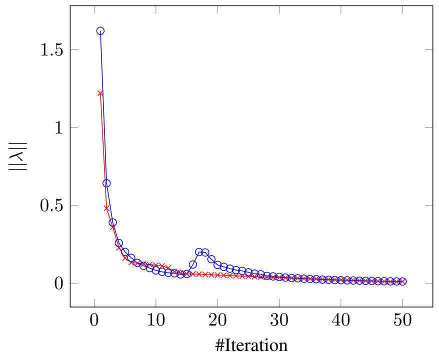

Figure 5. Convergence of the proposed DITML method and ITML on the ImageNet dataset. The iteration number here means the times that algorithms iterate over all the constraints.

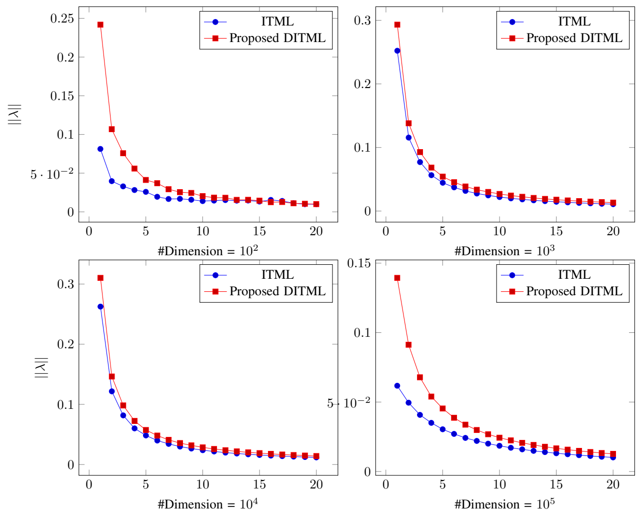

Figure 6. Convergence of ITML and the proposed DITML algorithms with different scaled synthetic dataset. The iteration number here means the times that algorithms iterate over all the constraints.

Acknowledgements

We would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper. The work described in this paper was partially supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 413213 of the General Research Fund) and 2015 Microsoft Research Asia Collaborative Research Program (Project No. FY16-RES-THEME-005).

	k-NN	ITML + k-NN	Proposed DITML + k-NN
Synthetic-10^2	0.903	0.932	0.924
Synthetic-10^3	0.940	0.962	0.957
Synthetic-10^4	0.933	0.938	0.938
Synthetic-10^5	0.812	0.923	0.900
ImageNet	0.682	0.835	0.830

Table I
Accuracy of k-NN classification when $k = 4$

References

[1] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 12, pp. 2143–2157, 2009.

[2] T. Kato and N. Nagano, "Metric learning for enzyme active-site search," *Bioinformatics*, vol. 26, no. 21, pp. 2698–2704, 2010.

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

[4] B. Kulis, "Metric learning: A survey," *Foundations and Trends in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2012.

[5] E. P. Xing, M. I. Jordan, S. Russell, and A. Y. Ng, "Distance metric learning with application to clustering with side-information," in *Advances in neural information processing systems*, 2002, pp. 505–512.

[6] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *Journal of Machine Learning Research*, vol. 10, pp. 207–244, 2009.

[7] M. Der and L. K. Saul, "Latent coincidence analysis: A hidden variable model for distance metric learning," in *Advances in Neural Information Processing Systems*, 2012, pp. 3230–3238.

[8] C. Shen, J. Kim, L. Wang, and A. Van Den Hengel, "Positive semidefinite metric learning using boosting-like algorithms," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1007–1036, 2012.

[9] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," in *Advances in Neural Information Processing Systems*, 2003.

[10] J. V. Davis and I. S. Dhillon, "Structured metric learning for high dimensional problems," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, 2008, pp. 195–203. [Online]. Available: http://doi.acm.org/10.1145/1401890.1401918

[11] G. Qi, J. Tang, Z. Zha, T. Chua, and H. Zhang, "An efficient sparse metric learning in high-dimensional space via l_1-penalized log-determinant regularization," in *ICML*, 2009, pp. 841–848. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553482

[12] A. Globerson and S. T. Roweis, "Metric learning by collapsing classes," in *Advances in Neural Information Processing Systems*, 2005.

[13] A. Agarwal and J. C. Duchi, "Distributed delayed stochastic optimization," in *Advances in Neural Information Processing Systems*, 2011, pp. 873–881.

[14] J. N. Tsitsiklis, D. P. Bertsekas, M. Athans *et al.*, "Distributed asynchronous deterministic and stochastic gradient optimization algorithms," *IEEE transactions on automatic control*, vol. 31, no. 9, pp. 803–812, 1986.

[15] M. Zinkevich, M. Weimer, A. J. Smola, and L. Li, "Parallelized stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2010, pp. 2595–2603.

[16] A. I. Chen and A. E. Ozdaglar, "A fast distributed proximal-gradient method," in *2012 50th Annual Allerton Conference on Communication, Control, and Computing, Allerton Park & Retreat Center, Monticello, IL, USA, October 1-5, 2012*, 2012, pp. 601–608. [Online]. Available: http://dx.doi.org/10.1109/Allerton.2012.6483273

[17] M. Li, D. G. Andersen, and A. Smola, "Distributed delayed proximal gradient methods," in *NIPS Workshop on Optimization for Machine Learning*, 2013.

[18] S. Sra, "Scalable nonconvex inexact proximal splitting," in *Advances in Neural Information Processing Systems*, 2012, pp. 539–547.

[19] J. Li, X. Lin, X. Rui, Y. Rui, and D. Tao, "A distributed approach toward discriminative distance metric learning," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 9, pp. 2111–2122, Sept 2015.

[20] L. M. Bregman, "The relaxation method of finding the common point of convex sets and its application to the solution of problems in convex programming," *USSR computational mathematics and mathematical physics*, vol. 7, no. 3, pp. 200–217, 1967.

[21] E. Fetaya and S. Ullman, "Learning local invariant mahalanobis distances," *arXiv preprint arXiv:1502.01176*, 2015.

[22] D. Lim, G. Lanckriet, and B. McFee, "Robust structural metric learning," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 615–623.

[23] A. Bellet, A. Habrard, and M. Sebban, "A survey on metric learning for feature vectors and structured data," *arXiv preprint arXiv:1306.6709*, 2013.

[24] C. Xiong, D. Johnson, R. Xu, and J. J. Corso, "Random forests for metric learning with implicit pairwise position dependence," in *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '12. New York, NY, USA: ACM, 2012, pp. 958–966. [Online]. Available: http://doi.acm.org/10.1145/2339530.2339680

[25] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," *Advances in neural information processing systems (NIPS)*, p. 41, 2004.

[26] Q. Qian, R. Jin, S. Zhu, and Y. Lin, "Fine-grained visual categorization via multi-stage metric learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3716–3724.

[27] P. Xie and E. Xing, "Large scale distributed distance metric learning," *arXiv preprint arXiv:1412.5949*, 2014.

[28] Y. Ying and P. Li, "Distance metric learning with eigenvalue optimization," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1–26, 2012.

[29] M. Frank and P. Wolfe, "An algorithm for quadratic programming," *Naval research logistics quarterly*, vol. 3, no. 1-2, pp. 95–110, 1956.

[30] J. Chen, T. Yang, and S. Zhu, "Efficient low-rank stochastic gradient descent methods for solving semidefinite programs," in *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, 2014, pp. 122–130.

[31] Y. Ma and T. Zheng, "Boosted sparse non-linear distance metric learning," *arXiv preprint arXiv:1512.03396*, 2015.

[32] J. Dhillon, "Differential entropic clustering of multivariate gaussians," *Advances in Neural Information Processing Systems*, vol. 19, p. 337, 2007.

[33] B. Kulis, M. A. Sustik, and I. S. Dhillon, "Low-rank kernel learning with bregman matrix divergences," *Journal of Machine Learning Research*, vol. 10, pp. 341–376, 2009.

[34] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2011, pp. 693–701.

[35] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, 2010, p. 10.

[36] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin, "Parallel coordinate descent for l1-regularized loss minimization," *arXiv preprint arXiv:1105.5379*, 2011.

[37] J. Langford, A. Smola, and M. Zinkevich, "Slow learners are fast," *arXiv preprint arXiv:0911.0491*, 2009.

[38] S. Shalev-Shwartz and Y. Singer, "Logarithmic regret algorithms for strongly convex repeated games," *The Hebrew University*, 2007.

[39] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of The 31st International Conference on Machine Learning*, 2014, pp. 647–655.