Distance Metric Learning Through Convex Optimization

David Alvarez-Melis DA1142@CIMS.NYU.EDU
Courant Institute
New York University
New York, NY 10012, USA

Abstract

We present a survey of recent work on the problem of learning a distance metric in the framework of semidefinite programming (SDP). Along with a brief theoretical background on convex optimization and distance metrics, we present various methods developed in this context under different approaches and provide theoretical analysis for a subset of them. A gradient ascent projection algorithm (Xing, 2002) and an approximate Frank-Wolfe method (Ying, 2012) are implemented and tested on several standard classification tasks from machine learning. We provide a comparison of the results obtained by our implementations, along with the corresponding results for some state-of-the-art algorithms.

1. Introduction

In several areas of machine learning the type of metric used has a central role on the performance of algorithms [12]. This is particularly true for problems, both supervised and unsupervised, where a certain measure of *similarity* between examples is required, such as in classification, clustering and k-nearest neighbors (kNN). The canonical distance measure, the Euclidean, is practical if no particular information on the source or target space is available. However, it fails to extract specific features of the inputs and thus is in disadvantage in front other problem-specific distance metrics which adapt to the nature of training examples. Indeed, extensive work on the subject has been done recently, and several authors have demonstrated improvements in various tasks by introducing non-euclidean metrics learnt from data (Shalev-Shwartz et al., 2004; Goldberger et al., 2005; Ying and Li, 2012).

Most approaches to the problem of learning a metric fall into two categories; those based on convex optimization and those that make use of eigenvector-style methods. Among the latter, we find well-known procedures from statistics that date back to the early 19th century, such as Pearson's Principal Component Analysis (PCA) and Fisher's Linear Discriminant Analysis (LDA). There has also been, however, recent development on this framework, for example with the introduction of Relevant Component Analysis (Shental et al. 2002; Bar-Hillel et al. 2003). This method seeks to amplify relevant variability in the data by applying an appropriate transformation, and does so by

making use of information shared among so-called "chunklets" ("small sets of data points, in which the class label is constant, but unknow" [8]).

The other class of methods, those rooted in optimization, set up the problem as a convex problem (usually an SDP) and then make use of various algorithms for these type of setting. This new approach to the problem, first proposed in the seminal work by Xing et al. (2002), set off a series of publications offering various frameworks to tackle this problem (Shalev-Shwartz et al., 2004; Weinberger et al., 2006; Weinberger and Saul, 2008). An overview of the various methods developed in this context is shown in Figure 1.

In this work, we restrict out attention to this optimization-based approach to distant metric learning. After providing the reader with the basic concepts behind metric learning and semidefinite programming, we review some of the theoretical settings and algorithms proposed in the literature. We focus particularly on two of these methods: one of the current state-of-the-art algorithms, Large Margin Nearest Neighbor Classification (Weinberger et al., 2006) and a novel formulation of the task as an eigenvalue optimization problem (Ying and Li, 2012). In Section (ref) we show our implementation of two of these methods, and in Section (ref) we present their results when tesetd on standard machine learning tasks, and compare their performance with state-of-the art methods. The main purpose of this survey is to provide a general overview of recent developments in the use of optimization for distance metric learning, and to point the interested reader in the direction of literature to explore the topic further.

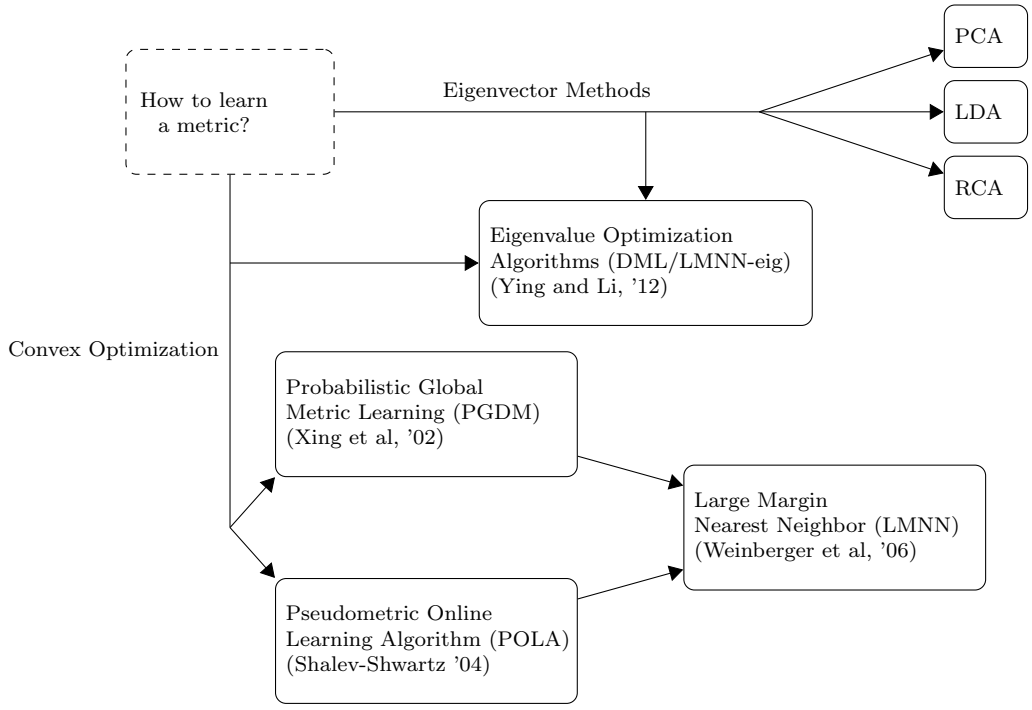

Figure 1: A general overview of the approaches to distance metric learning.

2. Distance Metric Learning

A metric is a fundamental tool to associate elements in a vector space, for it provides us with a notion of distance or "similarity" between those elements. Recall that if X is a $\mathbb{K}$ - vector space, the mapping $d : X \times X \to \mathbb{R}$ is a **metric** if it satisfies:[1]:

(i) $d(x, y) = d(y, x) \quad \forall x, y \in X$

(ii) $d(x, y) \leq d(x, z) + d(z, y) \quad \forall x, y, z \in X$

(iii) $d(x, y) = 0 \Leftrightarrow x = y$

(iv) $d(x, y) \geq 0$

In the definition above, if the condition (iii) is dropped, then the mapping d is said to be a pseudometric. Also, note that multiple metrics can be defined for the same space.

In inner product spaces, a natural way to define a metric is through the the inner product, and thus d can be defined by $d(x, y)^2 = \|x - y\|^2 = \langle x - y, x - y \rangle$. For general inner products, we can thus define a distance metric as

$$d_M(x, y)^2 = (x - y)^T M (x - y) \tag{1}$$

It is easy so see that me must require that M be strict and positive definite (or semidefinite) if d_M is to be a metric (pseudometric, respectively). Pseudometrics defined in this way are called *Mahalanobis distances*, and they will be the central part of our discussion. We will follow the convention of denoting $\mathbb{S}^d$ the space of symmetric $d \times d$ matrices, and $\mathbb{S}^d_+$ its subset of positive semidefinite (PSD) elements. Two observations are due at this point. First, note that by setting $M = I$ we can recover the Euclidean distance. Also, note that the fact that M is symmetric positive semidefinite means that it can be decomposed into its Cholesky factors as $M = LL^T$, and thus we can alternatively compute distances with it as $d_L(x, y) = \|L(x - y)\|_2^2$, where in this case $\|\cdot\|$ is the usual 2-norm. As pointed out by Weinberger and Saul (2009), this suggests two approaches for learning a distance metric: through the matrix M or L. Although the optimization problem in the latter case would be unconstrained (since there are no restrictions on L, as opposed to M required to be positive semidefinite), using M as our variable of interest allows for the setting of the problem as an SDP, which will offer various advantages.

Now, suppose that we have a set of training points $\{x_i : i \in \mathbb{N}\}$ and we are given a some notion of similarity between them. This information can be in the form of labels $\{y_i : i \in \mathbb{N}\}$ (in the case of supervised learning), or, more generally, as a similarity set $\mathcal{S}$, where $(i, j) \in \mathcal{S}$ if x_i and x_j are "similar". The dissimilarity set $\mathcal{D}$ is defined analogously. Information given in this form, without having explicit labels, is referred to as *side-information.* What criterion should be imposed on the M-distance so that

1. Formally, the non-negativity property is redundant in this definition, since the conditions (i), (ii) and (iii) imply that $0 = d(x, x) \leq d(x, y) + d(y, x) = 2d(x, y)$ for any pair (x, y).

this similarity information is captured? Naturally, this can be enforced by requiring that the distance between similar pairs is minimized, or equivalently, as in Xing et al. (2002), that the distance between dissimilar pairs is maximized, while similar pairs are kept close "enough". A further constraint is naturally to require that M be PSD. Thus, an appropriate distance metric can be learnt by solving the problem

$$\begin{aligned} \max_{M} \quad & \sum_{(i,j)\in\mathcal{D}} d(x_i, x_j)_M \\ \text{s.t.} \quad & \sum_{(i,j)\in\mathcal{S}} d(x_i, x_j)_M^2 \leq 1, \\ & M \succeq 0 \end{aligned} \tag{2}$$

It is easy to see that with this setting, the problem is convex. Therefore, efficient local-free-minima algorithms can be used to solve it.

3. Semidefinite Programming

Problem (2) belongs to a particular subfamily of convex optimization problems, namely that of Semidefinite Programms (SDP). These are concerned with the optimization of linear objective functions over the intersection of the cone of positive semidefinite matrices and a spectrahedron (the equivalent of a simplex in $\mathbb{R}^{n\times n}$). A general SDP has the form

$$\begin{aligned} \min_{X} \quad & C \bullet X \\ \text{s.t.} \quad & A_i \bullet X \qquad i = 1, \ldots, m, \\ & X \succeq 0 \end{aligned} \tag{3}$$

where $X \succeq 0$ means X is PSD and $U \bullet V = tr(U^T V)$ is the inner product between matrices. However, the are many equivalent formulations of an SDP, and the type used frequently depends on the particular characteristics of the problem. One of the other widely used standard representations of SDP is through the an eigenvalue optimization problem

$$\begin{aligned} \min_{x} \quad & t \\ \text{s.t.} \quad & tI - A(x) \geq 0 \end{aligned} \tag{4}$$

where $A(X) = A_0 + \sum x_i A_i$. In fact, eigenvalue optimization problems make a large part of the contexts where SDP arise, and have been researched extensively. One particular problem, that of minimizing the maximal eigenvalue of symmetric matrices (Lewis and Overton 1996) will pay particular importance in the formulation of section 4.3. Semidefinite programs also play a very useful role in combinatorial optimization, such as in the `MAX−CUT`, `SPARSEST−CUT` and `MIN−UNCUT` problems [1].

Most algorithms for solving SDP are based on primal-dual interior-point methods, and, although they have worst case complexities that grow as $O(n^{\frac{1}{2}}$, they tend to perform much better than that in practice [9]. Many features in interior-point methods for

solving linear programs are shared by the corresponding SDP variants. And, although highly dense matrix can make SDP particularly costly to solve, sparse problems and those with special structure can be solved much more efficiently[2].

4. Solving the Optimization Problem

In this section we will analyze some approaches to solving the distance metric learning problem (2). We will focus on three of them in particular; the original method to learn Mahalanobis distances proposed by Xing et al. (2002), the Large Margin Nearest Neighbor Classification algorithm due to Weinberger et al. (2009) and the recent formulation as an eigenvalue optimization problem due to Ying and Li (2012). The first and last of these methods are general in their nature: they learn a metric in that can be later used for clustering by k-means or nearest neighbor (kNN) classification. The second one, although specific to the context of kNN is of particular theoretical interest, and for this reason we include its analysis in this section.

4.1 Probabilistic Global Distance Metric Learning

In the original approach to the problem, Xing et al. dealt directly with the problem in its form (2). They define

$$g(M) = \sum_{(i,j)\in\mathcal{D}} d(x_i, x_j)_M \tag{5}$$

$$f(M) = \sum_{(i,j)\in\mathcal{S}} d(x_i, x_j)^2_M \leq 1 \tag{6}$$

and come up with a gradient ascent iterative method to solve optimize $g(M)$, where in each successive iteration feasibility is ensured by projecting the matrices iterates onto the sets $\mathcal{P} = \{M : \sum_{(i,j)\in\mathcal{S}} d(x_i, x_j)^2_M \leq 1\}$ and $\mathbb{S}^d_+$. Thus, the method consists of the projection steps

$$\tilde{M} = \arg\min_{M'} \{\|M' - M\|_F : M' \in \mathcal{P}\} \tag{7}$$

$$\hat{M} = \arg\min_{M'} \{\|M' - \tilde{M}\|_F : M' \in \mathbb{S}^d_+\} \tag{8}$$

iterated until convergence, followed by the gradient ascent step

$$M_{k+1} = \hat{M} + \alpha(\nabla_M g(\hat{M}))$$

Note that (7) is a Quadratic Programming (QP) problem with a single linear constraint, where the variable is the vectorized M, that is, $u_M = (M_{11}, M_{12}, \ldots, M_{1d}, M_{21}, \ldots, M_{dd})$.

For the subproblem (8), the authors propose to "force" positive semidefinitentess into the matrix M by discarding negative eigenvalues. Specifically, they obtain a (full)

2. Various references on these topics can be consulted in http://www.stanford.edu/ boyd/papers/pdf/semidefprog.pdf

eigendecomposition $M = X^T \Lambda X$ and then set $\Lambda' = \text{diag}(\max\{0, \lambda_1\}, \ldots, \max\{0, \lambda_1\})$. This step, although conceptually simple, turns out to be the weakest feature of this method, for the computation of full eigendecompositions is considerably expensive.

All combined, the gradient-ascent method proposed by Xing takes can be implemented with the following pseudocode.

Algorithm 1 Gradient ascent with iterative projection algorithm.

Input: $\mu > 0$, tol, α_t
  Initialize $S_1^\mu \in S_+^d$ with $tr(S_1^\mu) = 1$
  while **do**
    while **do**
      $M := \arg\min_{M'}\{\|M' - M\|_F : M' \in C_1\}$
      $M := \arg\min_{M'}\{\|M' - M\|_F : M' \in C_2\}$
    end while
    $M := M + \alpha(\nabla_M g(M))_{\perp \nabla_M f}$
  end while
  return M

4.2 Large Margin Nearest Neighbor Classification

An interesting generalization due to Weinberger et al. [11] of the arguments described above for distance metric learning combines ideas form various preceding methods, such as PGDM, POLA and NCA, by building on their individual strengths. Their method is specifically designed for the context of kNN classification. According to the authors, the key features of their method are (i) its convex loss function, (ii) its goal of margin maximization and (iii) kNN-specific constraints.

Within the context of kNN, the authors define the concept of *target neighbors*; those elements x_j that we require to be closest, under the learnt metric, to a particular element x_i. This relation is denoted by $i \rightsquigarrow j$. *Impostors* are those elements that invade (i.e. have a different label) the perimeters within which we enclose target neighbors. Thus, the goal of learning in this case is to maintain a large distance between impostors and the perimeters established by target neighbors. The method is made robust by maintaining a safety margin around the kNN decision boundaries. From this idea, the method takes its name, Largest Margin Nearest Neighbor Classification.

The loss function consists of two terms, one of which "pulls" target neighbors closer together, and one which "pushes" different neighbors apart. The former, shown in (9), is simply a reformulation of the original constraint in (2), for it penalizes large distances between target neighbors.

$$\epsilon_{\text{pull}}(M) = \sum_{j \rightsquigarrow i} \|x_i - x_j\|_M^2 \tag{9}$$

The second component of the loss function is

$$\epsilon_{\text{push}}(M) = \sum_{i,j \rightsquigarrow i} \sum_{l} \big[1 - I(y_i = y_l)\big] \big[1 + \|x_i - x_j\|_M^2 - \|x_i - x_l\|_M^2\big]_+ \tag{10}$$

where $[z]_+ = max(z, 0)$. This term penalizes small distances between elements with different labels, by controlling the violation of the margin inequality $\|x_i - x_l\|_M^2 \leq \|x_i - x_j\|_M^2 + 1$, where $i \rightsquigarrow j$ and x_l is an impostor. The final loss function is obtained by a weighted combination of the previous two ϵ-functions (9) and (9) as follows

$$\epsilon(M) = (1 - \mu)\epsilon_{\text{pull}}(M) + \mu\epsilon_{\text{push}}(M) \tag{11}$$

In the original formulation of the authors (in which distances are expressed in terms of L instead of $M = L^T L$, the objective function is not convex. Although a gradient descent method could be used to solve it, this approach will be vulnerable to running into local optima. Therefore, the authors transform their original problem into an SDP, first, by transforming adopting M as the variable of interests (as we have done in (11)) and requiring that M be SDP. In addition, to deal with the hinge loss function $[z]_+$, slack variables are introduced. For every triplet of target neighbors ($j \rightsquigarrow i$) and impostors x_l a nonnegative variable ξ_{ijl} is added, whose purpose is to measure the amount in which the margin constraint is violated. Finally, the problem takes the form

$$\begin{aligned} \underset{M}{\text{Minimize}} \quad & (1 - \mu) \sum_{i,j \rightsquigarrow i} \|x_i - x_j\|_M^2 + \mu \sum_{i,j \rightsquigarrow i,l} (1 - y_{il})\xi_{ijl} \\ \text{subject to} \quad & (1) \quad \|x_i - x_l\|_M^2 - \|x_i - x_j\|_M^2 \geq 1 + \xi_{ijl} \\ & (2) \quad \xi_{ijl} \geq 0 \\ & (3) \quad M \succeq 0 \end{aligned} \tag{12}$$

The authors use their own algorithm for solving this problem and achieve remarkable results, especially when preprocessing the data with PCA. A key difference between LMNN and PGDM as done in the previous section, is that LMNN learns local features of the metric, whilst the latter attempts to learn metrics globally. This accounts for a higher accuracy rate in most datasets, particularly in high-dimensional ones (Weinberger, 2009).

4.3 Eigenvalue Optimization

In recent work by Ying and Li [13], yet another approach to distance metric learning is proposed. The authors establish an equivalent min-max formulation of the problem of learning a distance metric, and then use an approximate Frank-Wolfe algorithm to solve it.

4.3.1 Reformulation as an Eigenvalue Problem

To recast (2) as an eigenvalue problem, the authors start by defining the simplex

$$\Delta = \{u \in \mathbb{R}^D : u_\tau \geq 0, \quad \sum_{\tau \in \mathcal{D}} u_\tau = 1\}$$

and the spectrahedron

$$\mathcal{P} = \{M \in \mathbb{S}_+^d : tr(M) = 1\}$$

To minimize the notation, we will adopt the authors' convention of denoting by $\tau = (i, j)$ the dissimilarity pairs and by $X_{ij} = (x_i - x_j)(x_i - x_j)^T$ the vector difference rank-one matrices. Then, the linear combination of the similarity matrices is given by $X_{\mathcal{S}} := \sum_{(i,j)\in\mathcal{S}} X_{ij}$.

Their main result shows that the original problem can be reformulated as

$$\min_{u\in\Delta} \max_{M\in\mathcal{P}} \left\langle \sum_{\tau\in\mathcal{D}} u_\tau \tilde{X}_\tau, M \right\rangle = \min_{u\in\Delta} \lambda_{max} \left(\sum_{\tau\in\mathcal{D}} u_t \tilde{X}_\tau \right) \tag{13}$$

where $\tilde{X}_\tau = X_{\mathcal{S}}^{-\frac{1}{2}} X_\tau X_{\mathcal{S}}^{-\frac{1}{2}}$. The convexity of Δ and $\mathcal{P}$ allows us to use a variant of Sion's mini-max Theorem to exchange the order of the operators above and obtain the equivalent problem

$$\max_{M\in\mathcal{P}} f(M) = \max_{M\in\mathcal{P}} \min_{u\in\Delta} \sum_{\tau\in\mathcal{D}} u_\tau \left\langle \tilde{X}_\tau, M \right\rangle \tag{14}$$

Furthermore, the authors introduce a smoothed version of problem (14), by defining

$$f_\mu(M) = \min_{u\in\Delta} \sum_{\tau\in\mathcal{D}} u_\tau \left\langle \tilde{X}_\tau, M \right\rangle + \mu \sum_{\tau\in\mathcal{D}} u_\tau \ln u_\tau = -\mu \ln \left(\sum_{\tau\in\mathcal{D}} e^{-\frac{\langle \tilde{X}_\tau, M\rangle}{\mu}} \right)$$

Thus, the problem of learning a metric can be cast as

$$\max_{M\in\mathcal{P}} f_\mu(M) \tag{15}$$

This is a linearly constrained convex optimization problem.

4.3.2 The Frank-Wolfe Algorithm

In 1956, Marguerite Frank and Philip Wolfe [3] proposed an algorithm for solving concave[3] quadratic programming problems with linear constraints, which they described as a "gradient-and-interoplation" method. This algorithm, also known as the *conditional gradient method*, solves in each iteration a subproblem given by a linear approximation of the objective function, subject to the original constraints, namely

$$\begin{aligned} \underset{x}{\text{Minimize}} \quad & z_k(y) = f(x_k) + \nabla f(x_k)^T (y - x_k) \\ \text{subject to} \quad & y \in S \end{aligned} \tag{16}$$

The solution y_k to this linear problem determines a search direction $p_k = y_k - x_k$. The convexity of the simplex S ensures that p_k is a feasible direction, and furthermore, it is easy to prove that it is a descent direction. The next step consists of finding a step size $0 \leq \alpha_k \leq 1$ such that $f(x_k + \alpha p_k)$ is minimized.

3. Or convex, with the appropriate change of sign.

Although appealing for its simplicity, the Frank-Wolfe method has only sublinear convergence in general and tends to slow down significantly when approaching the solution. Thus it is rarely used is practice for constrained optimization problems, unless its purpose is to obtain only an approximate solution. However, its suitability for high dimensional problems and the sparse convexity properties of its iterates have caused revived interest recently, particularly in the context of machine learning and transportation network problems.

Recently [4], Hazan extended the classic Frank-Wolfe algorithm to deal with SDP over the cone of positive semi-definite matrices with trace equal to one (that is, on the spectrahedron $\mathcal{P}$ defined in the previous section). Over the space of matrices, the original linear subproblem (16) is now a an eigenvalue problem, and is solved approximately in Hazan's algorithm. This method produces *sparse* solutions, where sparsity is understood in this context as a low-rank solution matrix. This property (through a Cholesky decomposition) makes the method particularly appealing in our machine learning context, where the high-dimensionality of the problems can be tackled with sparse matrix-vector products.

4.3.3 The DML-eig Algorithm

With the eigenvalue formulation given in section 4.3.1 and the revisited Frank-Wolfe algorithm of section 4.3.2 at hand, Ying and Li propose the following algorithm for learning a distance metric.

Algorithm 2 Approximate Frank-Wolfe Algorithm for Distance Metric Learning

Input: $\mu > 0$, tol, α_t
  Initialize $M_1^\mu \in \mathbb{S}_+^d$ with $tr(M_1^\mu) = 1$
  while $|f_\mu(M_{t+1}^\mu) - f_\mu(M_t^\mu)| > tol$ **do**
    Find $v =$eigmax$\nabla f_\mu(M_t^\mu)$
    $Z_t^\mu = vv^T$
    $M_{t+1}^\mu = (1 - \alpha_t)M_t^\mu + \alpha_t Z_t^\mu$
  end while
  return M_t^μ

Under certain conditions on the sequence of step sizes α_t used (namely, if $\alpha_t \to 0$ but $\sum \alpha_t$ diverges), it can be proven that the sequence of matrices M_t^μ generated by Algorithm 2 converges to the maximum of $f_\mu(M)$. In fact, for the particular sequence carefully chosen sequence $\{\alpha_t \propto t^{-1} : t \in \mathbb{N}\}$, the authors obtain the the bounds given by the following Theorem.

Theorem 1 *For any $0 < \mu \leq 1$, let $\{M_t^\mu : t \in \mathbb{N}\}$ be generated by Algorithm 2 with step sizes given by $\{\alpha_t = \frac{2}{t+1} : t \in \mathbb{N}\}$. Then, for any $t \in \mathbb{N}$ we have that*

$$\max_{M \in \mathcal{P}} f(M) - f_\mu(M_t^\mu) \leq 2\mu \ln D + \frac{8 \max_{\tau \in \mathcal{D}} \|\tilde{X}_\tau\|^2}{\mu t} + \frac{8 \ln D}{t}$$

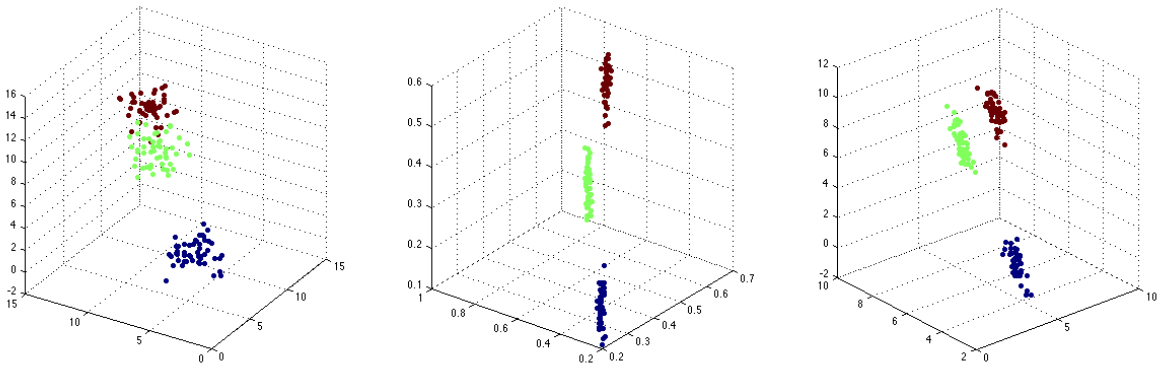

Figure 2: (a) Randomly sampled data, with classes indicated by colors. (b) Rescaled according to learnt metric from PGDM and (c) form DML-eig.

Thus, Algorithm 2 finds an ϵ-approximate solution to (15) in $O(d^2/\epsilon^2)$ (compare this to the $O(n^3)$ eigedecomposition needed in the algorithm of Section 2).

5. Experiments

We implemented and tested the two most general methods analyzed in this survey: the gradient ascent algorithm with projections (PGDM) of Section 2 by Xing et al. (2002) and the approximate Frank-Wolfe method (DML-eig) for the eigenvalue optimization problem by Ying and Li (2012) of Section 4.3.3. Our main goal was to compare the performance and complexity that result of setting the problem of learning a distance metric (2) in two considerably different theoretical frameworks, namely, that of a standard SDP problem (2) or a min-max eigenvalue optimization problem (14).

For the implementation of PGDM, we made use of Xing's auxiliary functions[4] for computing the value of f and its gradient, but coded the main method ourselves. The implementation of DML-eig was built from scratch. Additionally, we coded a k-means clustering method for MATLAB, largely based on Naothi Seos implementation[5], but modified to accept general Mahalanobis distances and not only covariance matrices. This, after being fed with the corresponding learnt metric matrix, was the main tool used for classification.

First, we analyze the distorting effects of the metrics learnt by these two methods on an artificial dataset. We created random points drawn from k normal multivariate distributions (where point from different distributions are considered to belong to different classes), and then used PGDM and DML-eig to train metrics M_{PGDM} and $M_{DML-EIG}$. Then, we rescale the sample points by the transformation $x \mapsto M^{\frac{1}{2}}$.

4. Available at http://www.cs.cmu.edu/epxing/papers/Old_papers/code_Metric_online.tar.gz
5. Code available at http://note.sonots.com/SciSoftware/kmeans.html

Figure 2 shows the different effect that the two metrics have on the data. As expected, the two methods have "clustered" points belonging to the same class, while separating different classes. Thus, the effect of the metrics can be interpreted as shrinking dimensions in which the points on the clusters are similar, and expanding those in which they are not. However, it is clear that the metric form DML-eig has done a more successful job of compressing similar points together. It is interesting how PGDM projects the points into plane, thus effectively reducing the dimension of the range of the clusters.

Now, we test our methods on two real classification tasks. The first one is Fisher's famous Iris flower data set. It consists of four features measured in 50 samples from each of the three species of the Iris flower (*Setosa, Virginica* and *Versicolor*). This classic data set is one the canonical classification tasks.

We ran 10 trials, randomly splitting in each trial the database in a 70/30% proportion for training and testing set, respectively. That is, the metrics were trained with 70% of the points, and then used to cluster (by using k-means) the 30% test set. Since we want to measure the performance in terms of similar pairs being grouped into the same class (regardless of which class it is), we need a special accuracy measure. We use the following, as suggested by Xing (2002):

$$\text{Accuracy} = \sum_{i>j} \frac{I\big(I(c_i = c_j) = I(\hat{c}_i = \hat{c}_j)\big)}{0.5m(m-1)}$$

where $I(\cdot)$ is the indicator function and m is the number of clusters (three in this case). The average results of the trials are shown on Table 1.

Table 1: Performance of clustering in the Iris dataset.

Metric	Euclidean	PGDM	DML-eig
Accuracy	84.3430	86.3961	89.7536

As expected, clustering with learnt matrices delivers better accuracy than with the "zero-information" identity (Euclidean) metric. Also, note that DML-eig performs considerably better than the other two. It is interesting to compare the metrics learnt in any given iteration by the two methods.

```
M_PGDM =

    0.1010    0.0004    0.0004   -0.0000
    0.0004    0.1005    0.0004    0.0002
    0.0004    0.0004    0.0998   -0.0003
   -0.0000    0.0002   -0.0003    0.1000
```

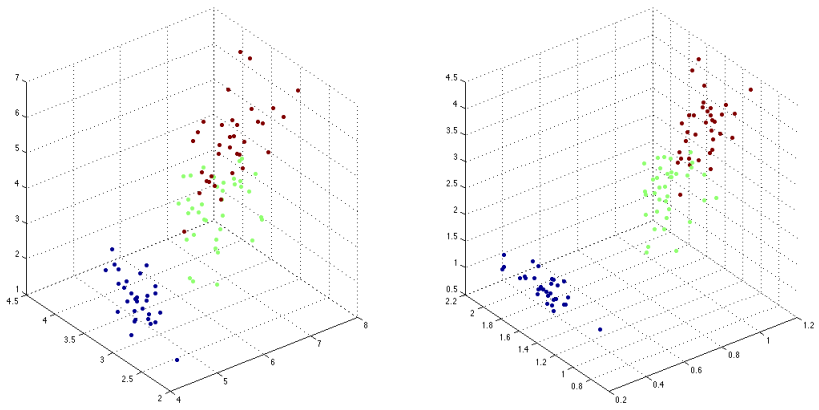

Figure 3: (a) Original test points. (b) Test points after metric transformation.

```
M_DMLEIG =

    0.0429   −0.0351    0.0285    0.0976
   −0.0351    0.3057   −0.0171   −0.0156
    0.0285   −0.0171    0.3755   −0.0514
    0.0976   −0.0156   −0.0514    0.2759
```

The matrices are certainly different, and there is obvious relation between them. This portrays the fact that the matrices are learnt through very different processes, neither of which yields a clear interpretation in terms of the actual data.

In Figure 3 we show (by plotting only on the first three coordinates of the space) the effect of distorting the test points by $M_{DML-eig}^{1/2}$. The increased accuracy of the method is easy to appreciate in this case: points from different classes which were before mixed are now almost linearly separable. Indeed, for this particular trial, the accuracy obtained after clustering with this metric was 95.17% against only 73.72% when using euclidean distances.

The second dataset in which we trained metrics was from a Breast cancer diagnosis task[6]. This base, containing entries with 107 features, was chosen to illustrate the performance of the two methods on problems with high-dimensionality. We made a subset of the data base containing subsamples with $10,000$, $50,000$ and $80,000$ entries. As before, we split these each time into a training and a testing set, learnt metrics on the former and clustered the latter by using k-mean with the learnt metric. The results are shown in Table 2.

We can see from these results that even though clustering with either metric was, again, better than using euclidean distances, this time DML-eig clearly outperformed

6. The database was used for a yearly data mining competition KDD Cup 2008. It can be obtained at www.kdd2008.com

Table 2: Performance of clustering in the Breast Cancer dataset.

Metric	Euclidean	PGDM	DML-eig
$10,000$	64.26	65.38	72.71
$50,000$	70.12	72.14	81.90
$80,000$	73.12	–	84.22

PGDM and was significantly better than using the identity as a metric. In the largest problem, PGDM failed to finish in a reasonable time. We now compare the running times of learning the metric by each of the two optimization methods. The same 2.53GHz Intel Core 2 Duo with 8Gb memory was used in all the trials.

Table 3: Running time (in seconds) of metric learning in the Breast Cancer dataset.

Sample Size	$10,000$	$50,000$	$80,000$
PGDM	$1,226$	$5,835$	–
DML-eig	448	$1,613$	$4,097$

The difference between the two results shown by 3 is considerable. PGDM consistently required more computational time. This gap between the methods' efficiency was amplified as the dimensionality of the training set used increased. With $80,0000$ entries, the PGDM did not finish and had to be interrupted. This poor performance in a large database is certainly largely due to the full eigendecomposition required by the first of these methods, as opposed to the $O(n^2)$ approximate algorithm used by the second one. We conclude that Ying and Li's DML-eig method provides better results for clustering, and is significantly more efficient that then corresponding PGDM (Xing et al, 2002) method.

6. Conclusions

We have analyzed in depth the problem of learning a distance metric from a sample of points of which we have a notion of similarity. We presented various approaches to this problem that can be found in the literature, most of which are fairly recent. We justified their derivations, presented convergence properties and briefly discussed their algorithmic implementation. In addition, we implemented two of the main methods and compared their performance on some classification tasks. This was used to exemplify the advances in solving this problem that has been made over the last few years.

Further work could be aimed at comparing the performance of optimization-based methods for DML to that of canonical statistical eigenvector methods, such as PCA and LDA. Another extension would be to investigate improvements to Ying and Li's framework by means of kernelization and linear transformations, which has been recently shown to be especially appropriate for high-dimensional data (Chatpatanasiri

et al., 2010; Jain et al. 2012). The list of approaches presented here is by no means exhaustive, and is intended only to provide the reader with a brief panorama of current research directions in the field.

List of Algorithms

List of Figures

References

[1] Sanjeev Arora and Satyen Kale, *A combinatorial, primal-dual approach to semidefinite programs*, in In Proceedings of the thirty-ninth annual ACM symposium on Theory of Computing, ACM Press, 2007, pp. 227–236.

[2] R. Chatpatanasiri, T. Korssrilabutr, P. Tangchanachaianan, and B. Kijsirkul, *A new kernelization framework for mahalanobis distance learning algorithms*, Neurocomputing, 73 (2010), pp. 1570–1579.

[3] Marguerite Frank and Philip Wolfe, *An algorithm for qudratic programming*, Naval Research Logistics, 3 (1956), pp. 95–110.

[4] Elad Hazan, *Sparse approximate solutions to semidefinite programs*, in Proceedings of the 8th Latin American conference on Theoretical Information, Springer-Verlag, 2008, pp. 306–316.

[5] Prateek Jain, Brian Kulis, Jason V. Davis, and Inderjit S. Dhillon, *Metric and kernel learning using a linear transformation*, J. Mach. Learn. Res., 13 (2012), pp. 519–547.

[6] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer, 2nd ed., 2006.

[7] Shai Shalev-Shwartz, Yoram Singer, and Andrew Ng, *Online and batch learning of pseudo-metrics*, in Proceedings of the Twenty First International Conference on Machine Learning (ICML-04), 2004, pp. 94–101.

[8] Noam Shental, Tomer Hertz, Daphna Weinshall, and Mish Pavel, *Adjustment learning and relevant component analysis*, in Proceedings of the Seventh European Conference on Computer Vision (ECCV-2002), vol. 4, Springer-Verlag, 2002, pp. 776–790.

[9] Lieven Vandenberghe and Stephen Boyd, *Semidefinite programming*, SIAM Review, 38 (1996), pp. 49–95.

[10] Kilian Q. Weinberger, J. Blitzer, and Lawrence K. Saul, *Distance metric learning for large margin nearest neighbor classification*, in Advances in Neural Information Processing Systems 18, MIT Press, 2005.

[11] Kilian Q. Weinberger and Lawrence K. Saul, *Distance metric learning for large margin nearest neighbor classification*, J. Mach. Learn. Res., 10 (2009), pp. 207–244.

[12] Eric Xing, Andrew Ng, Michael Jordan, and Stuart Russell, *Distance metric learning, with application to clustering with side-information*, in Advances in Neural Information Processing Systems 15, MIT Press, 2002, pp. 505–512.

[13] Yiming Ying and Peng Li, *Distance metric learning with eigenvalue optimzation*, J. Mach. Learn. Res., 13 (2012), pp. 1–26.

Main MATLAB routines used.

```
function [ M_DML,M_FW ] = MetricLearn(X,Y )
%Main function to perform distance metric learning, computed with ...
   two methods. X is a matrix of input instances and Y are their ...
   labels.

[n,d]= size(X);
[X_S,D,S] = createSD( X,Y );


%==== Two methods to obtain the distance metric matrix ...
   ====================

tic
% Xing's DML
M_0 = 0.1*eye(d);
w=X_S(:);
t = w'*M_0(:)/100;
[M_DML, converged] = iter_projection_new2(X, S, D, M_0, w, t, 100)
if(converged)
    fprintf('Metric learnt with Xings method.\n');
else
    fprintf('Xings method failed. No metric learnt for this ...
       algorithm.\n');
end
toc

tic
%Ying's Frank-Wolfe
Δ = 0.01;
X_S = X_S + Δ*eye(length(X_S));   %% For FW, Ridge to ensure PosDef
L = chol(X_S,'lower');
M_FW = FrankWolfe(X,D,L);
fprintf('Metric learnt with Frank-Wolfe Algorithm.\n');
toc

% ...
   ================================================================================


end

function [X_S,D,S] = createSD( X,Y )
%Create Similarity, Disimilarity and X_S matrix of Xings paper
[n,d]=size(X);
X_S = zeros(d);
D = zeros(n);
S = zeros(n);
k_d= 1; k_s=1;
for i=1:(n-1)
```

```
    for j=i+1:n
        if (Y(i) == Y(j))    %% They are similar
            S(i,j)=1;
            k_s=k_s+1;
            x_i = X(i,:);
            x_j = X(j,:);
            X_S = X_S + (x_i-x_j)'*(x_i-x_j);
        else     %% Dissimilar
            %             Col_Diss = L
            %             D(:,k) = (X(:,i) - X(:,j));
            D(i,j)=1;  %% Dissimilarity pairs
            k_d=k_d+1;
        end
    end
end
end


function [ M ] = FrankWolfe(X,D,L)
% FrankWolfe distance metric learning, as done in Ying and Li 2012
% INPUTS: S is a PSD Matrix for initialization, with t(S)=1
% OUTPUTS: M is a PSD Metric learnt from the data

mu = 10^(-5);
tol = 10^(-5);

% Choosing step sizes
theta = 1;

% Initialization
t=1;
fun = 0;
M = eye(size(X,2));

[newfun,grad] = funEval(M,X,D,L,mu);
while(   abs(newfun-fun)>tol   )
%     fprintf('Frank-Wolfe Iteration number %6d\n',t);
    fun = newfun;
    opts.issym=1;
    [v,lambda] = eigs(grad,1,'lm',opts);
    Z = v*v';
    alpha = t^(-theta);
    M = (1-alpha)*M + alpha*Z;
    [newfun,grad] = funEval(M,X,D,L,mu);
    t = t + 1;
end

end


function [fun,grad] = funEval(M,X,D,L,mu)
```

```
num=0; denom=0;
for i=1:size(D,1)
    for j = 1:size(D,1)
        if(D(i,j)==1)
            L_inv = L \(X(i,:)-X(j,:))' ;
            X_hat = L_inv*L_inv';
            exp_t = exp(-trace(X_hat'*M)/mu);
            num = num + exp_t*X_hat;
            denom = denom + exp_t;
        end
    end
end
grad = num/denom;
fun = -mu*log(denom);
end


function [M, converged] = DMLEIG(X, S, D, M, w, t, maxiter)
% DML-eig as done by Xing 2002

s = size(X);
N = s(1);
d = s(2);
error1=1e10; error2=1e10;
threshold2 = 0.01;
epsilon = 0.01;
maxcount = 100;

w1 = w/norm(w);
t1 = t/norm(w);

count=1;
alpha = 0.1;          % initial step size along gradient

grad1 = fS1(X, S, M, N, d);   % gradient of similarity constraint ...
   function
grad2 = fD1(X, D, M, N, d);   % gradient of dissimilarity ...
   constraint func.
G = grad_projection(grad1, grad2, d); % gradient of fD1 orthognal ...
   to fS1


A_last = A;         % initial M
done = 0;

while (~done)

 % projection of constrants C1 and C2 ------------------------------
 % -----------------------------------------------------------------
 M_cycle=count;
 projection_iters = 0;
```

```
satisfy=0;

while projection_iters < maxiter & ¬satisfy

  A0 = A;
  x0= A0(:);
  if w' * x0 ≤ t
    A = A0;
  else
    x = x0 + (t1—w1'*x0)*w1;
    A = vec2mat(x, N);
  end

  fDC1 = w'*x;
  A_1 = A;
  A = (A + A')/2;  % enforce A to be symmetric
  [V,L] = eig(A);  % V is an othornomal matrix of A's eigenvectors,
          % L is the diagnal matrix of A's eigenvalues,
  L = max(L, 0);
  A = V*L*V';

  fDC2 = w'*unroll(A);
  A_2 = A;       % resulting A from constraint 2

  % ...
      ----------------------------------------------------------------

  error2 = (fDC2—t)/t;
  projection_iters = projection_iters + 1;

  if error2 > epsilon
    satisfy=0;
  else
    satisfy=1;   % loop until constrait is not violated after ...
       both projections
  end

end  % end projection on C1 and C2

%[fDC1 fDC2]
%[error1, error2]


 % third constraint: Gradient ascent
 obj_previous = fD(X, D, A_last, N, d);
 obj = fD(X, D, A, N, d);

 if  (obj > obj_previous | count == 1) & (satisfy ==1)
   alpha =  alpha * 1.05;  A_last = A;
   grad2 = fS1(X, S, A, N, d);
   grad1 = fD1(X, D, A, N, d);
   M = grad_projection(grad1, grad2, d);
```

```
    A = A + alpha*M;
  else
    alpha = alpha/2;
    A = A_last + alpha*M;
  end;

  Δ = norm(alpha*M, 'fro')/norm(A_last, 'fro');
  count = count + 1;
  if count == maxcount | Δ <threshold2,
    done = 1;
  end;

end;


if Δ > threshold2,
  converged=0;
else
  converged=1;
end;


function [Cluster Codebook] = cvKmeans(X, K, M, stopIter, verbose)
% cvKmeans — K—means clustering
%
% Synopsis
%   [Cluster Codebook] = cvKmeans(X, K, [stopIter], [distFunc], ...
    [verbose])
%
% Description
%   K—means clustering
%
% Inputs ([]s are optional)
%   (matrix) X        D x N matrix representing feature vectors ...
    by columns
%                     where D is the number of dimensions and N ...
    is the
%                     number of vectors.
%   (scalar) K        The number of clusters.
%   (scalar) [stopIter = .05]
%                     A scalar between [0, 1]. Stop iterations if the
%                     improved rate is less than this threshold ...
    value.
%   (func)   [distFunc = @cvEucdist]
%                     A function handle for distance measure. The ...
    function
%                     must have two arguments for matrix X and Y. See
%                     cvEucdist.m (Euclidean distance) as a ...
    reference.
%   (bool)   [verbose = false]
%                     Show progress or not.
```

```
%
% Authors
%  Original Code by Naotoshi Seo <sonots(at)sonots.com>, Modified ...
   By David
%   Alvarez-Melis
if ¬exist('stopIter', 'var') || isempty(stopIter)
    stopIter = .05;
end
if ¬exist('distFunc', 'var') || isempty(distFunc)
    distFunc = @cvEucdist;
end
if ¬exist('verbose', 'var') || isempty(verbose)
    verbose = false;
end
[N D] = size(X);
if K > N,
    error('K must be less than or equal to the number of vectors N');
end
if ¬exist('M','var') || isempty(M)
    M = eye(size(X,2));
end


% Initial centroids
Codebook = X(randsample(N, K),:);

improvedRatio = Inf;
distortion = Inf;
iter = 0;
while true
    % Calculate Mahalanobis distances between each sample and ...
       each centroid
    d = MahalnobisDistance(Codebook, X,M);
    % Assign each sample to the nearest codeword (centroid)
    [dataNearClusterDist, Cluster] = min(d, [], 1);
    % distortion. If centroids are unchanged, distortion is also ...
       unchanged.
    % smaller distortion is better
    old_distortion = distortion;
    distortion = mean(dataNearClusterDist);

    % If no more improved, break;
    improvedRatio = 1 - (distortion / old_distortion);
    if verbose
        fprintf('%d: improved ratio = %f\n', iter, improvedRatio);
    end
    iter = iter + 1;
    if improvedRatio ≤ stopIter, break, end;

    % Renew Codebook
    for i=1:K
        % Get the id of samples which were clusterd into cluster i.
```

```
        idx = find(Cluster == i);
        % Calculate centroid of each cluter, and replace Codebook
        Codebook( i,:) = mean(X( idx,:),1);
    end
end
    Codebook = Codebook';

end

function dist = MahalnobisDistance(X, Y, M)
% cvMahaldist - Mahalanobis distance
[n d] = size(X);
[p d] = size(Y);
for i=1:n
    diff = repmat(X(i,:), p,1) - Y;
    dsq(i,:) = sum((M*diff').*diff' , 1);
end
dist = sqrt(dsq);
end
```