Foundations and Trends® in
Machine Learning
Vol. 5, No. 4 (2012) 287–364

DOI: 10.1561/2200000019

now
the essence of knowledge

Metric Learning: A Survey

By Brian Kulis

Contents

Foundations and Trends® in
Machine Learning
Vol. 5, No. 4 (2012) 287–364

DOI: 10.1561/2200000019

now
the essence of knowledge

Metric Learning: A Survey

Brian Kulis

Ohio State University, CSE Department, Columbus, OH 43210, USA,
kulis@cse.ohio-state.edu

Abstract

The *metric learning* problem is concerned with learning a distance function tuned to a particular task, and has been shown to be useful when used in conjunction with nearest-neighbor methods and other techniques that rely on distances or similarities. This survey presents an overview of existing research in metric learning, including recent progress on scaling to high-dimensional feature spaces and to data sets with an extremely large number of data points. A goal of the survey is to present as unified as possible a framework under which existing research on metric learning can be cast. The first part of the survey focuses on linear metric learning approaches, mainly concentrating on the class of Mahalanobis distance learning methods. We then discuss nonlinear metric learning approaches, focusing on the connections between the nonlinear and linear approaches. Finally, we discuss extensions of metric learning, as well as applications to a variety of problems in computer vision, text analysis, program analysis, and multimedia.

1

Introduction

Consider the images in Figure 1.1, and imagine a scenario in which we must compute similarity or distances over pairs of images (for example, for clustering or nearest neighbor classification). A basic question that arises is precisely *how* to assess the similarity or distance between the pairs of images. For instance, if our goal is to find matching faces based on identity, then we should choose a distance function that emphasizes appropriate features (hair color, ratios of distances between facial keypoints, etc). But we may also have an application where we want to determine the pose of an individual, and therefore require a distance function that captures pose similarity. Clearly other features are more applicable in this scenario. To handle multiple similarity or distance metrics, we could attempt to determine by hand an appropriate distance function for each task, by an appropriate choice of features and the combination of those features. However, this approach may require significant effort and may not be robust to changes in the data. A desirable alternative — and the focus of this survey — is to apply *metric learning*, which aims to automate this process and learn task-specific distance functions in a supervised manner.

Fig. 1.1 Example face data set. In one application, our notion of "distance" between faces may depend on the pose, whereas in another application it may depend on the identity.

A possible informal formulation of the metric learning problem could be given as follows: given an input distance function $d(\boldsymbol{x}, \boldsymbol{y})$ between objects $\boldsymbol{x}$ and $\boldsymbol{y}$ (for example, the Euclidean distance), along with supervised information regarding an ideal distance, construct a new distance function $\tilde{d}(\boldsymbol{x}, \boldsymbol{y})$ which is "better" than the original distance function (we could also easily replace "distance" with "similarity," and d with s for some similarity function $s(\boldsymbol{x}, \boldsymbol{y})$). This survey will focus, for the most part, on learning distance functions $\tilde{d}(\boldsymbol{x}, \boldsymbol{y})$ of the form $d(f(\boldsymbol{x}), f(\boldsymbol{y}))$ for some function f — that is, we learn some mapping f and utilize the original distance function over the mapped data. We will denote this approach as *global metric learning* methods, since they learn a single mapping f to be applied to all the data.

One possible drawback to the above definition of metric learning is that it assumes that we have at least some supervision available to learn the new distance; the fact that we assume supervision seems somewhat arbitrary. Take, for instance, dimensionality reduction: linear methods such as principal components analysis can be viewed as constructing a linear transformation P to be applied globally to the data,

in an unsupervised manner. The resulting distance between objects is therefore $d(P\boldsymbol{x}, P\boldsymbol{y})$, and one may claim that this is also a form of metric learning. In contrast, the methods we study typically have supervised information regarding the structure of the desired distance function. For example, one popular form of supervision — relative distance constraints — assumes we may not know the target distance between pairs of instances, but does assume we know that object $\boldsymbol{x}$ is more similar to $\boldsymbol{y}$ than it is to $\boldsymbol{z}$. The fact that the supervised information is a function of the ideal distance (or similarity) is key to distinguishing the methods we study in this survey from other existing techniques such as dimensionality reduction methods or classification techniques. Furthermore, incorporating such supervision lends itself to interesting algorithmic and analysis challenges, as we will see. Thus, in this survey we will mainly focus on metric learning as a supervised learning problem.

We will break down global metric learning into two subclasses — linear and nonlinear. For both cases, we will mainly focus on the case where the input distance function is the Euclidean distance, i.e., $d(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|_2$. In the linear case, we aim to learn a linear mapping based on supervision, which we can encode as a matrix G such that the learned distance is $\|G\boldsymbol{x} - G\boldsymbol{y}\|_2$. This paradigm is by far the most prevalent in the metric learning community due to the fact that many of the resulting formulations are tractable (at the least, local solutions can be found easily). To achieve convexity, many methods assume that G is square and full-rank, leading to convex optimization problems with positive semi-definiteness constraints. We will discuss such methods in Section 2.

We study nonlinear methods for global metric learning in Section 3. In this case, the distance function is the more general $d(\boldsymbol{x}, \boldsymbol{y}) = \|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_2$. One of the most well-understood and effective techniques for learning such nonlinear mappings is to extend linear methods via kernelization. The basic idea is to learn a linear mapping in the feature space of some potentially nonlinear function ϕ; that is, the distance function may be written $d(\boldsymbol{x}, \boldsymbol{y}) = \|G\phi(\boldsymbol{x}) - G\phi(\boldsymbol{y})\|_2$, where ϕ may be a nonlinear function. While it may not appear that we have gained anything by this, if we further assume that we can compute

the *kernel* function $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$, then it turns out that we may efficiently learn G in the input space using extensions of linear techniques. Crucially, the resulting algorithms scale independently of the dimensionality of the feature space of ϕ, allowing us to utilize kernel functions whose embedding functions may be extremely high-dimensional (or even infinite-dimensional, as in the Gaussian kernel). A core result that we discuss is a representer theorem that demonstrates when such metrics may be learned. Beyond kernelization, we discuss some other proposed methods for nonlinear metric learning, including methods based on neural networks.

The goal of the survey is to provide an overview of recent advances in metric learning. For the sake of clarity, we will attempt to present as much of the literature as possible under a unified framework. Of course, given the broad scope of the metric learning problem, and the fact that not all material fits neatly into such a unified presentation, we will have to divert from the main presentation from time to time. In addition to presenting the main metric learning models and algorithms that have been studied, we also focus on several recent applications, including applications from computer vision, multimedia, and text analysis. It is our hope that this survey will synthesize much of the recent work on metric learning, and inspire new algorithms and applications.

2

Distance Learning via Linear Transformations

We begin with the simplest and popular approach to learning metrics. This approach is often called "Mahalanobis metric learning" in the research community (though the metric learned is not the Mahalanobis distance, as we will discuss), and attempts to learn distances of the form $\|G\boldsymbol{x} - G\boldsymbol{y}\|_2$ for some matrix G.

2.1 A Simple Motivating Example

As an example, consider the "wine" data set from the UCI machine learning repository.[1] This data set contains 13 attributes obtained by chemical analyses of wines grown in Italy. The underlying classification problem is to determine to which of three classes of wines each instance belongs, based on these 13 attributes. Suppose we are interested in finding a distance function over the space of these attributes. The simplest and most obvious choice would be to compute the Euclidean distance between the attribute vectors corresponding to two wines. Unfortunately, a quick look at the underlying attributes shows why this will not

[1] Available at `http://archive.ics.uci.edu/ml/`

work. Nine of the attributes have an average attribute value across the data set in the range $[0, 10]$, 3 of the attributes have an average attribute value in the range $[10, 100]$, and one feature has an average value of 747. Computing the distance between two feature vectors is completely dominated by the single largest attribute, and therefore carries little information about the data as a whole. Further, if one uses the distance function in conjunction with a classifier such as a k-nearest neighbor classifier, the performance is poor.

This example highlights that, at the very least, appropriate *scaling* of the data features is necessary before applying a distance function such as the Euclidean distance. For instance, one option would be to normalize each attribute, or to whiten the data. Indeed, whitening the data and then applying the Euclidean distance is precisely the Mahalanobis distance. However, such unsupervised scaling is generally not sufficient: in one application, certain features may be more important than in other applications, and thus the weighting of the features may change from application to application. This argument suggests the need for supervised learning of feature weighting to transform the data from one space to another. In the context of metric learning, the goal will be to learn such transformations of the data based on supervised information regarding the *distances* of the transformed data.

Linear transformation methods for metric learning are simply methods for learning the weights for a scaling and rotation of the data, given appropriate supervised data. In the following sections, we will describe a formalization for learning linear transformations for metric learning, and then we will describe several special cases which have received particular attention in the machine learning community. In subsequent sections, we will see how to generalize the linear transformation model for learning nonlinear transformations.

2.2 Basic Techniques and Notation

We begin with some notation and comparison to basic existing methods. Let us suppose that we have a set of data points in a Euclidean space $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$. Let $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ be the matrix of all the data points. We will use the Euclidean distance $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$

$= \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^T(\boldsymbol{x}_i - \boldsymbol{x}_j)}$ as the canonical distance measure,[2] and the inner product $\boldsymbol{x}_i^T\boldsymbol{x}_j$ as the canonical inner product. Let $\boldsymbol{e}_i$ be the ith standard basis vector, i.e., a vector of all zeros except for entry i, which has a 1.

2.2.1 The Mahalanobis Distance

The Mahalanobis distance [51] is defined as the following:

$$d_{\text{Mahal}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^T\Sigma^{-1}(\boldsymbol{x}_i - \boldsymbol{x}_j)},$$

where Σ is the covariance matrix of the data. In many cases, the true covariance is unknown, and so a point estimate of the covariance (e.g., the sample covariance) is used. The Mahalanobis distance is closely related to data whitening. When whitening a random vector $\boldsymbol{w}$ whose mean and covariance are μ and Σ, respectively, we compute the whitened version $\tilde{\boldsymbol{w}}$ via

$$\tilde{\boldsymbol{w}} = \Sigma^{-1/2}(\boldsymbol{w} - \mu).$$

It is straightforward to show that the resulting random variable has mean zero and covariance equal to the identity. Further, the Euclidean distance between two whitened variables is simply the Mahalanobis distance:

$$\begin{aligned}
&\sqrt{(\tilde{\boldsymbol{w}}_i - \tilde{\boldsymbol{w}}_j)^T(\tilde{\boldsymbol{w}}_i - \tilde{\boldsymbol{w}}_j)} \\
&\quad= \sqrt{\begin{array}{l}(\Sigma^{-1/2}(\boldsymbol{w}_i - \mu) - \Sigma^{-1/2}(\boldsymbol{w}_j - \mu))^T \\ \qquad \times (\Sigma^{-1/2}(\boldsymbol{w}_i - \mu) - \Sigma^{-1/2}(\boldsymbol{w}_j - \mu))\end{array}} \\
&\quad= \sqrt{(\boldsymbol{w}_i - \boldsymbol{w}_j)^T\Sigma^{-1}(\boldsymbol{w}_i - \boldsymbol{w}_j)} = d_{\text{Mahal}}(\boldsymbol{w}_i, \boldsymbol{w}_j).
\end{aligned}$$

Note that, in the case of the wine example discussed earlier, the use of the Mahalanobis distance would avoid the scenario in which one feature dominates in the computation of the Euclidean distance, as the data has implicitly been whitened.

[2] In fact, we will most often work only with the square of the Euclidean distance, which is not a metric or pseudo-metric as it does not satisfy the triangle inequality. However, most of the time it is algorithmically simpler to work with the squared Euclidean distance.

In the metric learning literature, the term "Mahalanobis distance" is often used to denote any distance function of the form:

$$d_A(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T A(\boldsymbol{x} - \boldsymbol{y}),$$

where A is some positive semi-definite matrix (i.e., its eigenvalues are non-negative). As with the original Mahalanobis distance, we can view this distance simply as applying a linear transformation of the input data: since A is positive semi-definite, we factorize it as $A = G^T G$ and simple algebraic manipulations show that $d_A(\boldsymbol{x}, \boldsymbol{y}) = \|G\boldsymbol{x} - G\boldsymbol{y}\|_2^2$. Thus, this generalized notion of a Mahalanobis distance exactly captures the idea of learning a global linear transformation.

2.2.2 Unsupervised Metric Learning and Dimensionality Reduction

A number of classical dimensionality reduction methods may be viewed as Mahalanobis distance learning methods, though we will tend to not focus on such classical methods throughout the survey. For instance, consider principal components analysis (PCA) [60] (or its probabilistic extensions [56, 70]), which finds a linear transformation to map the data from an input space to a lower-dimensional space such that the lower-dimensional projected data is as informative as possible, in that it captures as much of the variance of the data as possible. Classical supervised methods that discover linear mappings in the data include linear discriminant analysis [53], which uses class labels to maximally separate two classes of data.

While these methods are indeed related to Mahalanobis distances and metric learning — they can be viewed as inducing a Mahalanobis distance obtained via the projections computed by their respective algorithms — our focus will be on methods that specifically target learning the underlying metric, and employ some form of supervision to do so.

2.3 Regularized Transformation Learning

We now turn to a formal model for global linear metric learning with the Euclidean distance. We propose a general, regularized model that captures most of the existing techniques studied previously in the

literature, and which has the added benefit of allowing for general analysis, which will be useful when discussing nonlinear generalizations.

2.3.1 Model

Recall our informal definition of metric learning: we aim to learn a new distance using supervision that is a function of the learned distance/similarity. In the case of global linear metric learning, the original distance is the squared Euclidean distance and the learned distance is the squared Euclidean distance after applying the transformation G globally.

We would like the class of supervision to be as general as possible but still allow for useful analysis. To that end, we will assume that the supervision depends on the data only through the *mapped inner product* matrix $X^T G^T G X = X^T A X$. This allows, for example, supervision based on the mapped Euclidean or squared Euclidean distance since $\|G\boldsymbol{x}_i - G\boldsymbol{x}_j\|_2^2 = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T A(\boldsymbol{x}_i - \boldsymbol{x}_j) = (\boldsymbol{e}_i - \boldsymbol{e}_j)^T X^T A X(\boldsymbol{e}_i - \boldsymbol{e}_j)$. Similarly, supervision based on mapped inner products is also permitted, as well as linear combinations of mapped distances or similarities. In Section 2.3.2, we will describe various types of supervision that are utilized by existing techniques.

To encode the supervision, we assume that we are given a collection of m loss functions, which we will denote as $c_1, \ldots, c_m$; by the assumption above, each loss function depends on the data only through the inner product matrix $X^T A X$. For instance, one loss function might encode the squared loss between the target distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and the squared Euclidean distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ using the Mahalanobis distance with A.

The second part of the model is a regularizer on the model, which we will denote as $r(A)$, and will be a function of A. Putting these two together, we obtain the general model, a linear combination between these two components of the model:

$$\mathcal{L}(A) = r(A) + \lambda \sum_{i=1}^{m} c_i(X^T A X). \tag{2.1}$$

The λ term is a trade-off between the regularizer and the loss. The goal will be to find the minimum of $\mathcal{L}(A)$ over the domain of A, which we

will denote as $\text{dom}(A)$. In many cases, the domain of A is defined as the space of positive semi-definite matrices, but in some cases we will want to further restrict the domain (for example, to the space of non-negative diagonal matrices). Note that the final form of the model is analogous to the standard regularized empirical risk minimization problem utilized for classification problems such as support vector machines [59], the lasso [69], or (regularized) logistic regression [4], except that the optimization is with respect to a matrix and the loss functions have a particular form.

Before proceeding to discuss some examples of this model, we make a few remarks. First, sometimes the metric learning model will be specified as a *constrained* optimization problem of the form:

$$\begin{aligned} &\min_{A \in \text{dom}(A)} && r(A) \\ &\text{subject to } && c_i(X^T A X) \leq 0, \quad 1 \leq i \leq m. \end{aligned}$$

In general, we can transform this constrained problem into the unconstrained model, but we will often present the constrained version when discussing models that were originally developed in this form.

Second, we could have specified the regularizer in terms of G instead of A, and constraints of the form $c_i(X^T G^T G X)$, and then posed the minimization problem with respect to G. Indeed, neighbourhood components analysis [26], discussed below, solves a problem of this form. In many cases, however, solving with respect to A yields *convex* models, whereas solving with respect to G does not. Recall that a convex optimization problem seeks to solve the following problem [6]:

$$\begin{aligned} &\min && f_0(\boldsymbol{y}) \\ &\text{subject to } && f_i(\boldsymbol{y}) \leq 0, \quad i = 1, \ldots, m \\ &&& \boldsymbol{a}_i^T \boldsymbol{y} = b_i, \quad i = 1, \ldots, p \end{aligned}$$

The functions $f_0, f_1, \ldots, f_m$ are all convex. Note that we may replace the inequalities with generalized inequalities [6]; for instance, if the optimization problem is defined with respect to a matrix Y, the constraint that Y be positive semi-definite is a convex constraint that may be incorporated into the optimization problem. Returning to the metric

learning problem, writing in terms of A instead of G often yields a convex optimization problem since the mapped inner product $\boldsymbol{x}_i^T A \boldsymbol{x}_j$ and squared Euclidean distance $(\boldsymbol{x}_i - \boldsymbol{x}_j)^T A(\boldsymbol{x}_i - \boldsymbol{x}_j)$ are linear with respect to A, whereas they are quadratic with respect to G. Combining this fact with the types of constraints that are typically used (and discussed below), the final form of the optimization problem is often convex with respect to A. It is often beneficial to work with convex optimization problems as they can be solved in polynomial time [6].

2.3.2 Examples of Regularizers and Constraints

Before discussing some specific examples of the above model, let us briefly consider regularization and supervision in a general setting. The two most popular forms of supervision for metric learning are given by (a) similarity/dissimilarity constraints, and (b) relative distance constraints. For similarity constraints, we are given a set of pairs $(i,j) \in \mathcal{S}$ of objects that should be similar, and pairs $(i,j) \in \mathcal{D}$ of objects that should be dissimilar. Typically, we want to ensure that

$$\begin{aligned} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) &\leq u \quad (i,j) \in \mathcal{S} \\ d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) &\geq \ell \quad (i,j) \in \mathcal{D}. \end{aligned}$$

We can encode these as loss functions in various ways; for example, the hinge loss would encode the desired constraints as:

$$\begin{aligned} c(X^T AY) &= \max(0, d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) - u), \quad (i,j) \in \mathcal{S} \\ c(X^T AY) &= \max(0, \ell - d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)), \quad (i,j) \in \mathcal{D}. \end{aligned}$$

Similarly, the squared hinge loss would encode the constraints as:

$$\begin{aligned} c(X^T AY) &= (\max(0, d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) - u))^2, \quad (i,j) \in \mathcal{S} \\ c(X^T AY) &= (\max(0, \ell - d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)))^2, \quad (i,j) \in \mathcal{D}. \end{aligned}$$

Note that the above losses rely on an appropriate choice of u and ℓ. Below, we will use the notation $[z]_+ = \max(0, z)$ for the standard hinge loss.

Another possibility is to utilize relative distances. Typically, these are specified via a triple $(i, j, k) \in \mathcal{R}$ which denotes that $\boldsymbol{x}_i$ should have a smaller distance to $\boldsymbol{x}_j$ than $\boldsymbol{x}_j$ to $\boldsymbol{x}_k$:

$$d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) < d_A(\boldsymbol{x}_i, \boldsymbol{x}_k).$$

Note that, unlike similarity and dissimilarity constraints, the relative distances do not require one to specify any parameters. However, typically one adds a *margin* to the above constraint:

$$d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) < d_A(\boldsymbol{x}_i, \boldsymbol{x}_k) - m,$$

for some m. In many cases, m is chosen to be 1. These can be encoded into loss functions analogously to the similarity and dissimilarity constraints. As we will see in the representative examples, other constraints are possible, and have been utilized in the literature. For instance, one approach is to constrain the sum of distances of pairs of similar objects to be bounded. However, the use of similarity/dissimilarity and relative distance constraints appears to be the most popular method of introducing supervision.

In many applications, the form of the side information is governed by the application. For instance, suppose we are applying metric learning for face identity, and we want to gather supervision from human subjects. It is typically easier for a subject to provide relative distance constraints than similarity and dissimilarity constraints (i.e., it is possible to say that image a is more similar to b than to c, but it may be difficult to determine whether an arbitrary pair of images should be considered similar or dissimilar). On the other hand, if we have a fully supervised training data set consisting of class labels for all training data, it is straightforward and standard to create similarity constraints for all pairs of objects of the same class and dissimilarity constraints for pairs of objects of different classes. In practice, the more constraints one gives (assuming appropriate regularization), the better the performance of the resulting metric is; as a result, for fully supervised cases one often uses all possible pairs. Note that one could also create relative distance constraints given a fully supervised training set, though the number of such constraints grows cubically with the size of the training set.

Regarding regularizers, we will motivate several cases in the examples below. However, to get a sense of some potential regularizers, each of the following have been considered:

$$r(A) = \frac{1}{2}\|A\|_F^2$$
$$r(A) = \mathrm{tr}(AC)$$
$$r(A) = \mathrm{tr}(A) - \log\det(A).$$

The particular choice of regularizer has a significant impact on the resulting metric learning model, both theoretically and algorithmically. For instance, when C is the identity matrix, the resulting regularizer is the trace-norm (or nuclear-norm) regularizer, which is known to prefer low-rank solutions. For the metric learning problem, this corresponds to finding matrices G which reduce the dimensionality of the input data, and therefore these methods can be viewed as a form of dimensionality reduction.

2.4 Representative Special Cases

We now describe, in more detail, several existing models for linear metric learning. As we will see, several of these methods can be very simply obtained by picking a regularizer and a form of constraints. However, the choice of regularizer and constraints have important implications for both algorithms and the properties of the resulting metrics.

2.4.1 Frobenius Norm Regularization: $r(A) = \|A\|_F^2$

One of the most popular techniques involves regularizing based on the squared Frobenius norm, i.e., $r(A) = \|A\|_F^2$. This regularizer may be viewed as the matrix analog of the standard squared-ℓ_2 regularizer that is used in problems such as support vector machines and ridge regression. As such, the squared Frobenius norm inherits several of the useful properties that make the squared ℓ_2 regularizer popular, such as its ease in analysis and strong convexity. We will begin our discussion of special cases of our general model with three existing techniques.

2.4.1.1 Schultz and Joachims

At a high level, the method of Schultz and Joachims [61] may be viewed as employing a squared Frobenius norm regularizer with relative distance constraints, under the additional assumption that the matrix learned is a diagonal matrix. Learning a diagonal Mahalanobis matrix is equivalent to simply learning weights on the features and evaluating the Euclidean distance over the re-weighting of the data points. One advantage to learning a diagonal Mahalanobis matrix is that the number of parameters only grows linearly with the number of data dimensions, and is therefore significantly more scalable than general Mahalanobis matrix learning methods. Thus, Schultz and Joachims learn feature weights such that the relative distance constraints are satisfied.

More formally, Schultz and Joachims consider learning a Mahalanobis matrix A of the form $A = \tilde{A}D\tilde{A}^T$, where D is a diagonal matrix and $\tilde{A}$ is a given (possibly non-diagonal) matrix. Then the optimization is expressed in terms of D as

$$\begin{gathered}\min_D \qquad \|A\|_F^2 + \lambda \sum_{i=1}^{m} c_i(X^T A X), \\ \text{where } c_i(X^T A X) = [1 + d_A(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}) - d_A(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_3})]_+, \ (i_1, i_2, i_3) \in \mathcal{R} \\ A = \tilde{A}D\tilde{A}^T \succeq 0, D \text{ diagonal.}\end{gathered}$$

Here $\mathcal{R}$ is a set of relative distance constraints, which are enforced through the hinge loss. In the case where $\tilde{A} = I$, then the optimization problem is a squared Frobenius regularized metric learning problem with an additional constraint that the Mahalanobis matrix is diagonal. The domain of A is the set of all positive semi-definite matrices of the form $\tilde{A}D\tilde{A}^T$.

Since we are equivalently learning a vector of feature weights, we see that the optimization problem involves a quadratic objective over the feature weights, along with the hinge loss applied to a linear function of the feature weights. Thus, the resulting optimization problem is quite similar to the classical soft-margin SVM problem [59].

2.4.1.2 Kwok and Tsang

In follow-up work, Kwok and Tsang [47] presented a method that uses Frobenius regularization and similarity/dissimilarity constraints, and places no diagonal matrix restriction on the form of A. Although their method is mainly presented as a technique for learning kernels, they also provide an interpretation of their approach as learning a Mahalanobis metric.

The form of constraints they utilize is slightly different from other methods. Consider a similarity constraint $d_A(\boldsymbol{x}, \boldsymbol{y}) \leq u$, for some $\boldsymbol{x}$ and $\boldsymbol{y}$ that should be "similar." In many cases, one simply chooses a global value for u that is used for all constraints. Instead, Kwok and Tsang propose that the constraint should be $d_A(\boldsymbol{x}, \boldsymbol{y}) \leq d_I(\boldsymbol{x}, \boldsymbol{y})$; that is, the learned distance should be smaller than the original distance. Note the similarity to a relative distance constraint; however, in this case $d_I(\boldsymbol{x}, \boldsymbol{y})$ is fixed upfront, and so we can view the constraint as a similarity constraint where the bound of the constraint depends on $\boldsymbol{x}$ and $\boldsymbol{y}$. They define a dissimilarity constraint as $d_A(\boldsymbol{x}, \boldsymbol{y}) \geq d_I(\boldsymbol{x}, \boldsymbol{y}) + \gamma$, which ensures that for points that are dissimilar, their distance decreases by at least γ. They refer to this technique as "idealizing" the distance.

As with Schultz and Joachims, the approach of [47] utilizes a hinge loss over the constraints. They further let γ be part of the optimization, and add a penalty within the optimization for γ. The final form of the optimization is given as:

$$\begin{aligned} \min_{A \succeq 0, \gamma \geq 0} \quad & \|A\|_F^2 + \sum_i c_i(X^T A X) - \lambda_g \gamma \\ \text{where} \quad & c_i(X^T A X) = \lambda_s [d_A(\boldsymbol{x}_i, \boldsymbol{y}_i) - d_I(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+, \quad (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{S} \\ & c_i(X^T A X) = \lambda_d [d_I(\boldsymbol{x}_i, \boldsymbol{y}_i) + \gamma - d_A(\boldsymbol{x}_i, \boldsymbol{y}_i)]_+, \quad (\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D} \end{aligned}$$

When written in this way, the above optimization does not quite fall under the regularization framework that we have adopted due to the γ term. However, for completeness we note that, with some additional effort, we can write this in our model by considering

$$\bar{A} = \begin{bmatrix} A & \mathbf{0} \\ \mathbf{0}^T & -\gamma \end{bmatrix} \tag{2.2}$$

as an "augmented" Mahalanobis matrix. We consider two augmentations of the data: $\bar{\boldsymbol{x}} = [\boldsymbol{x}^T\ 1]^T$ and $\hat{\boldsymbol{x}} = [\boldsymbol{x}^T\ 0]^T$ and we let the full augmented data set X have copies of each. Then $d_{\bar{A}}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{y}}) = d_A(\boldsymbol{x}, \boldsymbol{y}) - \gamma$ and $d_{\bar{A}}(\hat{\boldsymbol{x}}, \hat{\boldsymbol{y}}) = d_A(\boldsymbol{x}, \boldsymbol{y})$. We let the domain of $\bar{A}$ be the set of matrices of the form given in (2.2), where A is positive semi-definite and $\gamma \geq 0$, and we rewrite our optimization and constraints appropriately over $\bar{A}$. When rewriting the constraints, the similarity constraints utilize $\hat{\boldsymbol{x}}$ and dissimilarity constraints utilize $\bar{\boldsymbol{x}}$. Finally, we define the regularizer as $r(\bar{A}) = \|A\|_F^2 - \lambda_g \gamma$.

2.4.1.3 Pseudo-Metric Online Learning Algorithm (POLA)

The POLA algorithm of Shalev-Shwartz et al. [63] is an example of a Frobenius-regularized Mahalanobis metric learning method applied in the online setting. We will discuss further details of the algorithm below, and focus here only on the model.

The authors assume that each constraint is a similarity or dissimilarity constraint. Let $y_i = 1$ if the pair of points for constraint i, $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, should be similar, and $y_i = -1$ if the pair should be dissimilar. The following loss functions are used by POLA:

$$c_i(X^T A X) = [1 + y_i(d_A(\boldsymbol{x}_i, \boldsymbol{y}_i) - \gamma)]_+.$$

As with Kwok and Tsang, we optimize jointly for A and γ, and can use the same trick during optimization: we augment A by incorporating γ and padding appropriately with zeros. Finally, though the authors do not explicitly state it, the regularizer is the squared Frobenius norm, which follows from the fact that the algorithm employs Euclidean projections to define the updates. When we discuss the algorithm, we will make these connections more precise.

2.4.2 Linear Regularization: $r(A) = \mathrm{tr}(AC)$

We now consider several existing models where the regularizer is a linear function with respect to the Mahalanobis matrix A.

2.4.2.1 Mahalanobis Metric Learning for Clustering

One of the earliest techniques for Mahalanobis metric learning was proposed by Xing et al. [80], sometimes referred to as MMC. The main idea behind this method is to minimize the sum of distances that should be similar while maximizing the sum of distances that should be dissimilar. As with other methods based on similarity and dissimilarity constraints, we assume a set of similar pairs $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}$ and dissimilar pairs $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}$ are given. The optimization problem to solve is as follows:

$$\begin{aligned} &\min_{A \succeq 0} \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \\ &\text{such that} \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}} \sqrt{d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)} \geq 1. \end{aligned}$$

The authors utilize $\sqrt{d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)}$ instead of the usual (squared) Mahalanobis distance because the solution with the squared Mahalanobis distance results in a trivial, rank-one solution to the problem. They discuss both the diagonal case (when the domain of A is restricted to a positive semi-definite diagonal matrix), and the full case (when the domain of A is any positive semi-definite matrix). For the full case, they utilize a gradient-descent algorithm combined with a projection onto the cone of positive semi-definite matrices. We can view the above problem in terms of the constrained version of our metric learning model. Here, the regularizer can be written as $\text{tr}(AC)$, where $C = \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T$, along with a single constraint for the dissimilarity constraints. The main application of this work was in improving k-means-type clustering methods by appropriately learning the distance function prior to clustering.

Although it does not fall under the regularization framework that we have been discussing, we note the related method of Bilenko et al. [3], which also employs Mahalanobis metric learning for clustering. Briefly, the authors augment a constrained k-means formulation (a clustering objective with penalties based on must-link and cannot-link constraints) by further optimizing with respect to the Mahalanobis (inverse covariance) matrix of each cluster. Thus, the E-step computes the

cluster assignments, while the M-step updates the means and the covariance matrices for each cluster.

2.4.2.2 Large-Margin Nearest Neighbors (LMNN)

Weinberger et al. [76, 78] proposed a metric learning technique based on a combination of relative distance constraints and the regularizer of Xing et al. That is, the regularizer is the same as in the Xing method — we can write is as $\mathrm{tr}(AC)$, where $C = \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T$. Then, similarly to Schultz and Joachims, Weinberger et al. utilize relative distances. They denote this method as LMNN, which stands for large-margin nearest neighbors. This model is one of the most popular methods for metric learning, and we will discuss several extensions of the LMNN model throughout this survey. Written out, the aim is to minimize:

$$\min_{A \succeq 0} \sum_{(i,j) \in \mathcal{S}} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda \sum_{(i,j,k) \in \mathcal{R}} [1 + d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) - d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)]_+.$$

The intuition behind LMNN is fairly straightforward: the goal is that a given data point should share the same labels as its nearest neighbors, while data points of different labels should be far from the given point. The relative distance with margins enforce such an intuition, hence the name large-margin nearest neighbors. Figure 2.1 illustrates the idea behind LMNN; the term *target neighbor* refers to a point that

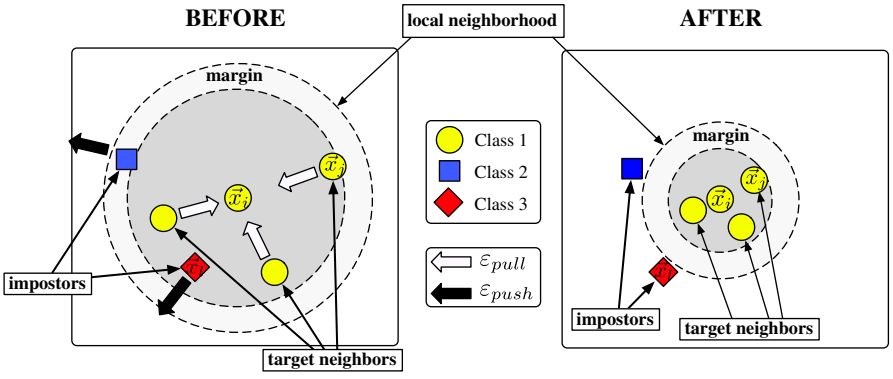

Fig. 2.1 Figure reproduced from [78] demonstrating the goals of LMNN. See text for details.

should be similar while an *impostor* is a point that is a nearest neighbor but has a different label. The goal of LMNN is to minimize the number of impostors via the relative distance constraints. The set $\mathcal{S}$ is defined as all pairs of target neighbors, which is often given by the set of pairs (i, j) where $\boldsymbol{x}_j$ is one of the k-nearest neighbors in the same class as $\boldsymbol{x}_i$; the set $\mathcal{R}$ is defined as all triples (i, j, k) such that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are target neighbors and $\boldsymbol{x}_k$ is a point with a different label.

2.4.2.3 Trace-norm Regularization

A final example of linear regularization arises when we choose $C = I$; in this case the resulting regularization is simply $r(A) = \mathrm{tr}(A)$. This is known as the trace-norm, or nuclear-norm.

The trace-norm is known to produce matrices A of low-rank; the regularizer is analogous to ℓ_1 regularization used for sparse linear models such as the lasso [69]. Since the resulting A matrix is typically low-rank, we can write $A = G^T G$, where $G \in R^{k \times d}$, with $k < d$. Therefore, these methods can be viewed as *supervised dimensionality reduction* methods. One particular method, studied in [36], utilizes the trace-norm regularizer with general linear constraints. If we choose similarity and dissimilarity constraints, we obtain the following (constrained) problem:

$$\begin{aligned} \min_{A \succeq 0} \quad & \mathrm{tr}(A) \\ \text{such that } & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \quad (i,j) \in \mathcal{S} \\ & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \quad (i,j) \in \mathcal{D} \end{aligned}$$

As will be discussed in the next section, the trace-norm formulation was originally described in conjunction with kernelization, in order to describe a supervised nonlinear dimensionality reduction scheme.

2.4.2.4 Neighbourhood Components Analysis (NCA) and Maximally Collapsing Metric Learning (MCML)

A slightly different formulation, called neighbourhood components analysis (NCA), was proposed by Goldberger et al. [26]. As we will

see, the resulting objective is non-convex and does not utilize a linear regularizer but its convex extension, MCML [25], does. Therefore, we will categorize this technique under the linear regularization heading.

If we let ℓ_i be the class associated with data point $\boldsymbol{x}_i$, then the objective proposed is simply:

$$\max_{A \succeq 0} \sum_{i=1}^{n} \sum_{\{j | \ell_i = \ell_j, i \neq j\}} \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))}.$$

The authors of NCA also consider the following alternate objective:

$$\max_{A \succeq 0} \sum_{i=1}^{n} \log \left(\sum_{\{j | \ell_i = \ell_j, i \neq j\}} \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))} \right).$$

The main idea behind these objectives is to optimize a softmax version of the leave-one-out KNN score. As opposed to the standard leave-one-out KNN score, which would just consider the distance to the nearest neighbor, NCA utilizes a continuous and differentiable relaxation of it, as given above. Although not particularly insightful, this objective may be viewed in the framework of the general linear metric learning model as an unconstrained problem where there is no regularizer, and n constraints of the form:

$$c_i(X^T A X) = \sum_{\{j | \ell_i = \ell_j, i \neq j\}} \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))}.$$

We define the constraints analogously for the alternate NCA objective. One drawback to the above objectives is that both are *non-convex* in A, and so a globally optimal solution cannot be efficiently found. Since the problem is non-convex, the authors of NCA instead solve the problem with respect to G, where as usual $A = G^T G$. They simply run gradient descent over G; we will discuss gradient descent in Section 2.5.

As follow-up work to NCA, Globerson and Roweis [25] proposed a variant to NCA which has the desirable property of convexity. It is based on a similar idea to NCA, except with a modification of the objective. To use the notation adopted in the MCML paper,

define the NCA objective as a set of conditional probabilities $p^A(j|i)$, defined as

$$p^A(j|i) = \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))},$$

along with an "ideal" conditional distribution

$$p_0(j|i) \propto \begin{cases} 1 & \ell_i = \ell_j, \\ 0 & \ell_i \neq \ell_j. \end{cases}$$

The NCA objective aims to find the matrix A that maximizes $p^A(j|i)$ summed over all i. In contrast, the MCML objective aims to minimize the KL-divergence between p_0 and p^A:

$$\min_{A \succeq 0} \sum_{i=1}^{n} \mathrm{KL}(p_0(j|i) | p^A(j|i)).$$

Now, using the fact that $\mathrm{KL}(p|q) = \sum_i p(i) \log(p(i)/q(i))$ and noting that $\sum_i p(i) \log p(i)$ is a constant, we can simplify the objective as follows:

$$\min_{A \succeq 0} \sum_{\{i,j|\ell_i = \ell_j\}} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) + \sum_{\{i,j|\ell_i = \ell_j\}} \log \left(\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)) \right).$$

The authors further assume that all classes are equally likely, and so we can write the resulting optimization as:

$$\min_{A \succeq 0} \mathrm{tr}(AC) + \lambda \sum_i \log \left(\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)) \right),$$

where $C = \sum_{\{i,j|\ell_i = \ell_j\}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T$; note the similarity to the method of Xing et al. as well as LMNN. The resulting objective is *convex* in A, since one can show that the log-sum-exp function is convex; further, each of the resulting log-sum-exp terms is clearly a function of the matrix $X^T A X$. We can therefore view the resulting optimization problem in our framework as utilizing a linear objective with convex penalties.

We also note the similarity between NCA and work on Informative Discriminant Analysis [41], a technique that learns a linear

transformation via gradient descent over an objective involving class-conditional distributions that are similar to those of NCA.

2.4.3 Information-Theoretic Metric Learning (ITML): $r(A) = \mathrm{tr}(A) - \log\det(A)$

The method of Davis et al., called information-theoretic metric learning (ITML) [20], considers the regularizer $r(A) = \mathrm{tr}(A) - \log\det(A)$. This regularizer can be viewed as a special case of the *LogDet divergence*, which is defined as

$$D_{\ell d}(A, A_0) = \mathrm{tr}(AA_0^{-1}) - \log\det(AA_0^{-1}) - d,$$

where d is the dimensionality of the data. Simple algebraic manipulations demonstrate that $D_{\ell d}(A, I) = r(A)$, and further that $D_{\ell d}(A, A_0) = r(A_0^{-1/2} A A_0^{-1/2})$.

The LogDet divergence has various properties that can be readily verified (see [46] for a detailed discussion), many of which are useful in metric learning contexts:

- *Scale invariance.* The divergence satisfies $D_{\ell d}(A, A_0) = D_{\ell d}(\alpha A, \alpha A_0)$, for $\alpha > 0$.
- *Translation invariance.* More generally, for any invertible S, the LogDet divergence satisfies $D_{\ell d}(A, A_0) = D_{\ell d}(S^T A S, S^T A_0 S)$.
- *Range space preservation.* While it appears that the definition of the LogDet requires the arguments to be strictly positive definite, one can show that $D_{\ell d}(A, A_0)$ is finite if and only iff $\mathrm{range}(A) = \mathrm{range}(A_0)$, and therefore the divergence can be defined for positive semi-definite matrices.
- *Connections to multivariate Gaussians.* Consider a multivariate Gaussian parameterized by mean $\boldsymbol{\mu}$ and precision matrix A: $p(\boldsymbol{x}; \boldsymbol{\mu}, A) = \frac{1}{Z}\exp(-d_A(\boldsymbol{x}, \boldsymbol{\mu}))$. Then for two multivariate Gaussians of the same mean, with precision matrices A_0 and A, we have:

$$KL(p(\boldsymbol{x}; \boldsymbol{\mu}, A_0) \| p(\boldsymbol{x}; \boldsymbol{\mu}, A)) = \frac{1}{2} D_{\ell d}(A, A_0),$$

where KL is the Kullback–Leibler divergence.

The name information-theoretic metric learning is derived from the last property, as the resulting metric learning optimization may be viewed in an information-theoretic manner. The LogDet divergence has been studied in the statistics community, where it is often referred to as Stein's loss [39]; the divergence also arises in the context of quasi-Newton optimization, where the updates of the BFGS and DFP algorithms can be given an interpretation of solving a LogDet optimization problem at each iteration [21].

The authors of ITML argue that the above properties, along with the fact that the LogDet divergence is only defined over the space of positive semi-definite matrices, make it a natural choice for regularization in a metric learning model. They therefore consider similarity and dissimilarity constraints under the above regularizer, and pose the learning problem as a constrained optimization problem:

$$\begin{aligned} \min_{A} \; & r(A) = \operatorname{tr}(A) - \log\det(A) \\ \text{such that } \; & d_A(\boldsymbol{x}, \boldsymbol{y}) \leq u, \quad (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S} \\ & d_A(\boldsymbol{x}, \boldsymbol{y}) \geq \ell, \quad (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}. \end{aligned}$$

One can write this problem as an unconstrained problem by incorporating slack variables; the authors of ITML consider one such formulation, and solve it using the method of Bregman projections. We will discuss this technique further in Section 2.5.3.

ITML was one of the first methods, along with POLA, for which extensions were performed from the linear metric learning problem to a nonlinear problem via kernelization. ITML and its variants (particularly LEGO [37] and HOLLER [68]) have also been extended to the online setting, and solved via stochastic gradient algorithms. We will also discuss these algorithms below, along with some of their associated bounds.

2.5 Optimization Techniques

Now that we have seen several examples of models proposed for learning Mahalanobis distances, we turn to the problem of efficiently optimizing the resulting models. In general, there is no single optimization technique that is universally accepted among the various models presented

earlier. Indeed, the authors of most existing models design their own optimization techniques specific to the individual models. These algorithms run the gamut across the range of possible methods, and a full treatment of such techniques is beyond the scope of this survey. We will summarize some of the main techniques that have been utilized successfully for metric learning applications. The development will be partially general and partially specific; in most cases we will discuss a particular model for which a given optimization procedure has been defined, but we will also discuss how to apply such techniques in a more general setting.

2.5.1 Gradient Descent

Perhaps the simplest optimization technique is standard gradient descent [6]. However, despite its simplicity, even gradient descent can be non-trivial to apply to the metric learning problem. This is because gradient descent is designed for unconstrained optimization problems — even in the "unconstrained" metric learning problem, there is still a constraint over the domain of the matrix. Typically the domain is simply the cone of positive (semi-)definite matrices, so even standard gradient descent cannot be naively applied.

One possible approach when the domain of A is the cone of positive semi-definite matrices, which was utilized for the NCA model, is to factorize the matrix $A = G^T G$, and then apply gradient descent on G directly. The positive semi-definiteness constraint over A implies that we can always factorize in this manner, so re-writing the optimization with respect to G allows us to remove the constraint. For instance, an optimization of the form:

$$\min_{A \succeq 0} \mathcal{L}(A) = r(A) + \lambda \sum_{i=1}^{m} c_i(X^T A X)$$

can simply be rewritten as an unconstrained optimization problem as follows:

$$\min_{G} \mathcal{L}(G) = r(G^T G) + \lambda \sum_{i=1}^{m} c_i(X^T G^T G X).$$

Gradient descent then proceeds by iteratively computing the gradient of $\mathcal{L}$ with respect to G and moving in the direction of the gradient:

$$G_{t+1} = G_t - \eta_t \nabla \mathcal{L}(G_t),$$

where η_t is the step size for iteration t. Unfortunately, this reformulation in terms of G comes at a price: if the original problem is convex in A, the new problem will almost always be non-convex in G. Nonetheless, applying gradient descent directly over G has two important advantages: (i) one can directly control the rank of A by choosing the size of G (to guarantee a low-rank solution, one can choose G to be $r \times d$, where $r < d$), (ii) the resulting algorithm is simple, fast, and scalable.

As an example, consider the model of NCA:

$$\max_{A \succeq 0} \sum_{i=1}^{n} \sum_{\{j|\ell_i=\ell_j, i\neq j\}} \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k\neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))}.$$

The model is already non-convex, so the introduction of the factorization does not eliminate convexity. Using the notation introduced in the discussion of MCML along with the shorthand $\boldsymbol{x}_{ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$, it is straightforward to show that the gradient with respect to G of the above model can be written as

$$-2G \sum_{\{i,j|\ell_i=\ell_j\}} p^A(j|i) \left(\boldsymbol{x}_{ij}\boldsymbol{x}_{ij}^T - \sum_k p^A(k|i)\boldsymbol{x}_{ik}\boldsymbol{x}_{ik}^T \right).$$

Given this gradient computation, one can directly apply standard unconstrained gradient-based optimization techniques (gradient descent, L-BFGS [2], etc).

2.5.2 Projected Gradient Descent

On the other hand, if our model is convex and we would like to retain convexity, we can apply the projected gradient method [27, 49] (which is a special case of a technique called generalized gradient descent). The basic idea is that, instead of applying just a gradient step, which would not retain the positive semi-definiteness of A in general, we apply

gradient descent followed by an orthogonal projection onto the positive semi-definite cone. That is, we repeat the following:

$$A_{t+\frac{1}{2}} = A_t - \eta_t \nabla \mathcal{L}(A_t)$$
$$A_{t+1} = \underset{A_*}{\operatorname{argmin}} \|A_* - A_{t+\frac{1}{2}}\|_F^2 \text{ s.t. } A_* \succeq 0$$

The first step is the standard gradient descent step, and the second step is the projection back to the cone of positive semi-definite matrices. It turns out that this projection is straightforward: we compute the eigendecomposition of $A_{t+\frac{1}{2}}$ and set all negative eigenvalues to 0. As in standard gradient descent, care must be taken to ensure that the step-sizes η_t are chosen appropriately so that one can guarantee convergence.

One example of the use of the projected gradient descent algorithm is the method utilized by Globerson and Roweis for MCML. Recall the MCML formulation:

$$\min_{A \succeq 0} \operatorname{tr}(AC) + \lambda \sum_i \log \left(\sum_{k \neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)) \right).$$

To compute the gradient steps, we first compute the gradient of the above regularized loss function. Then we apply gradient descent on A followed by an eigendecomposition where all negative eigenvalues are set to 0.

A related technique was utilized by Xing et al. for their metric learning problem. Recall the optimization problem to be solved:

$$\min_{A \succeq 0} \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)$$
$$\text{such that} \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{D}} \sqrt{d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)} \geq 1.$$

This formulation has 2 constraints: the positive semi-definiteness constraint, as well as the constraint over the sum of the dissimilar pairs. Thus, rather than project onto the single constraint $A \succeq 0$, we need to project onto the set which is the intersection of these two constraints. Projecting onto the intersection of these two constraints

can be performed by repeatedly projecting onto the two constraints individually, until convergence. For details, we refer the reader to the original paper [80].

2.5.3 Bregman Projections

In cases where there are a large number of constraints, it may be expensive to compute the entire gradient of the loss function. The next two techniques — Bregman projections and stochastic gradient descent — are based on making simpler updates based on a single constraint at a time. Both have been successfully utilized for optimization of several metric learning models.

The method of Bregman projections (also simply called Bregman's algorithm) is a simple first-order technique developed by Bregman in 1967 [8] for solving optimization problems where a strictly convex function must be minimized with respect to linear inequality constraints:

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(x) \\ \text{such that } & A\boldsymbol{x} \leq b. \end{aligned}$$

We can easily think of the basic ITML formulation in this manner, where we now consider the variable to optimize to be the matrix A, the function $f(A) = \text{tr}(A) - \log\det A$, and the constraints as either similarity and dissimilarity constraints or relative distance constraints (both of which are linear in the entries of A). We note again that positive definiteness is enforced automatically by the loss function, and is therefore not explicitly considered as a constraint.

The idea behind Bregman's algorithm is to choose one constraint at each iteration, and perform a projection so that the chosen constraint is satisfied. Unlike projected gradient descent, the Bregman projection is not in general an orthogonal projection, but is rather tailored to the particular function that is being optimized. After projecting, an appropriate correction is employed. Alternatively, we may view Bregman's algorithm simply as a dual coordinate ascent procedure, where we optimize the dual with respect to a single dual variable at each iteration. To be explicit, let us consider the projection step when choosing a single constraint, which we will write as $\text{tr}(AX_i) \leq b_i$. First, we consider the

equality constraint $\text{tr}(AX_i) = b_i$:

$$\begin{aligned} \min_A \quad & f(A) \\ \text{such that } & \text{tr}(AX_i) = b_i. \end{aligned}$$

If we introduce a dual variable α_i and form the Lagrangian $f(A) + \alpha_i(b_i - \text{tr}(AX_i))$, the Bregman projection is obtained by setting the gradient of the Lagrangian with respect to A and α_i to zero, resulting in the following system of equations:

$$\begin{aligned} \nabla f(A_{t+1}) &= \nabla f(A_t) + \alpha_i X_i \\ \text{tr}(A_{t+1} X_i) &= b_i. \end{aligned}$$

One must perform a correction when the constraint is an inequality constraint. Let $\nu_i \geq 0$ be a dual variable corresponding to the inequality constraint $\text{tr}(AX_i) \leq b_i$. After solving for α_i by solving for the Bregman projection as given above, we perform the following operations:

$$\alpha_i' = \min(\nu_i, \alpha_i), \quad \nu_i = \nu_i - \alpha_i'.$$

Finally, we update A_t to A_{t+1} by computing

$$\nabla f(A_{t+1}) = \nabla f(A_t) + \alpha_i' X_i.$$

Let us consider the ITML problem in the context of Bregman's algorithm. Recalling that $f(A) = \text{tr}(A) - \log\det A$, we see that $\nabla f(A) = I - A^{-1}$. For simplicity, let us consider a similarity constraint of the form $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u$, which we can write as $\text{tr}(AX) \leq u$, where $X = (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T$. The Bregman projection must solve the following system of equations:

$$\begin{aligned} A_{t+1}^{-1} &= A_t^{-1} - \alpha_i X \\ \text{tr}(A_{t+1} X) &= u. \end{aligned}$$

Since X is a rank-one matrix, we can utilize the Sherman-Morrison inverse formula [28]:

$$(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^T A^{-1}}{1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u}}.$$

It turns out that there is a simple closed-form solution to the resulting system of equations. Letting $\boldsymbol{z} = \boldsymbol{x}_i - \boldsymbol{x}_j$ and $p = \boldsymbol{z}^T A_t \boldsymbol{z}$, after simplification we arrive at a simple rank-one update: $A_{t+1} = A_t + \beta A_t \boldsymbol{z}\boldsymbol{z}^T A_t$, where $\beta = \alpha/(1 - \alpha p)$ and $\alpha = (1/p) - (1/u)$. It can easily be verified (Lemma 9, [46]) that the update results in a matrix A_{t+1} that remains in the positive semi-definite cone. Finally, one must perform the appropriate correction, which still results in a simple rank-one update to the matrix A.

In practice, there will often be no feasible solution to the general ITML problem, particularly when the number of constraints is large. It is straightforward to introduce *slack variables* into the metric learning formulation. Though we will not discuss this in detail, see [20] and Section 6.2.1 of [46] for further details on how to add slack variables to guarantee a feasible solution.

In the case of the function $f(A) = \mathrm{tr}(A) - \log\det A$, we need not worry about the positive semi-definiteness constraint, as the regularizer automatically enforces such a constraint. It may therefore appear that the use of Bregman's algorithm is somewhat limited, in that it can only handle linear constraints; when we use a regularizer such as $f(A) = \frac{1}{2}\|A\|_F^2$, the positive semi-definiteness must be explicitly enforced. While we cannot use Bregman's algorithm directly, an extension to Bregman's algorithm called Dykstra's algorithm can be employed in the more general setting. We will not discuss this algorithm in detail, but we refer the interested reader to [7] for further details.

2.5.4 Stochastic Gradient Descent and Online Learning

Consider the problem of minimizing the sum of the losses $\sum_i c_i(X^T A X)$. Analogous to projection-based methods, we may only want to update based on a single loss c_i at a time. Stochastic gradient descent (SGD) [5] and online learning methods provide a way to achieve this goal.

The basic SGD algorithm considers a single c_i per iteration and updates A via

$$A_{t+1} = A_t - \eta_t \nabla c_i(X^T A X).$$

Once again, the value η_t is the learning rate, analogous to gradient descent. Typically one performs a projection onto the PSD cone after performing this update, as in projected gradient descent. Compared with the gradient descent procedure described earlier, the main difference is that the SGD approach does not consider the gradient of the entire loss; further, the regularizer is not explicitly considered here.

We can alternately write this update as the solution to the following problem:

$$A_{t+1} = \operatorname*{argmin}_A \frac{1}{2}\|A - A_t\|_F^2 + \eta_t c_i(X^T A X).$$

In this view, we can say that we are trying to update the matrix A such that the updated matrix is "close" to A_t in terms of the squared Frobenius distance (as captured by the first term of the problem), while also trying to minimize the loss of the ith constraint (as captured by the second term of the problem). In the online learning literature, this is often referred to as a balance between conservativeness and correctiveness.

One can generalize the SGD approach by changing the notion of conservativeness; this is often called *mirror descent* [1], and can be expressed as

$$A_{t+1} = \operatorname*{argmin}_A D_\phi(A, A_t) + \eta_t c_i(X^T A X),$$

where $D_\phi(A, A_t)$ is an arbitrary Bregman divergence. As we will discuss below, changing the divergence implicitly changes the form of regularization.

Online learning algorithms are often analyzed in terms of a notion called *regret* which, loosely speaking, looks at the difference between the loss of a fixed predictor and the loss of the online predictions:

$$Reg = \sum_t c_t(X^T A_t X) - \sum_t c_t(X^T A^* X),$$

where c_t indexes the constraint examined at step t, and A^* is the matrix which minimizes the sum of the losses. Though it is beyond the scope of this manuscript to discuss online learning algorithms in detail, we note that there is a wide body of literature on online convex programming,

including [10, 84], and many of this literature can be applied to the design of competitive online metric learning algorithms. For example, choosing $\eta_t = 1/\sqrt{t}$ typically yields regret of $O(\sqrt{T})$, where T is the total number of timesteps/constraints.

Let us look at two examples of SGD/online learning methods for metric learning. The first is called LEGO [37], and is similar to the basic ITML algorithm. As with ITML, we consider similarity and dissimilarity constraints, which we encode via a squared hinge-loss. We then use mirror-descent, with the Bregman divergence arising from the regularizer for ITML: $f(A) = \mathrm{tr}(A) - \log\det A$. As discussed earlier, the resulting divergence is called the LogDet divergence: $D_{\ell d}(X, Y) = \mathrm{tr}(XY^{-1}) - \log\det(XY^{-1}) - d$, where d is the dimensionality of the data. The corresponding mirror descent problem can be expressed as:

$$A_{t+1} = \mathrm{argmax}_A D_{\ell d}(A, A_t) + \eta_t c_i(X^T A X).$$

For similarity and dissimilarity constraints, this update turns out to have a simple, rank-one solution which is similar to the update for ITML. We refer the reader to [37] for further details.

As a second example, consider POLA [63] from the perspective of online learning and stochastic gradient descent. If we let the learning rate η_t tend toward infinity, the SGD update becomes a projection: we find a matrix A that satisfies the chosen constraint while minimizing the divergence to the existing matrix. Recall the loss functions used by POLA:

$$c_i(X^T A X) = [1 + y_i(d_A(\boldsymbol{x}_i, \boldsymbol{y}_i) - \gamma)]_+.$$

With a sufficiently high learning rate, we can view the standard SGD update as an orthogonal projection onto the set of matrices A where the above loss is 0. As shown in [63], this results in a simple rank-one update to A as well as a simple update to γ. However, since we are utilizing the squared Frobenius norm (as opposed to the LogDet divergence), positive semi-definiteness is not enforced by the update. Thus, we must additionally project back onto the cone of positive semi-definite matrices. This is achieved by setting any negative eigenvalues of A to equal 0. (By the eigenvalue Interlacing Theorem [28], the rank-one update

to A from above causes at most one eigenvalue to become negative, and so at most eigenvalue must be set to 0, which can be performed efficiently.) Similarly, the update to γ is simply to set it to $\max(\gamma, 1)$.

The authors of POLA show regret-type bounds for the resulting procedure by adapting analysis for existing online convex programming algorithms such as the passive-aggressive online learning technique [15].

2.5.5 Complexity of Optimization

We will say a few words about the complexity of the different optimization procedures. We are mainly interested in the running time of the methods with respect to the number of dimensions of the input data as well as the number of constraints.

For gradient descent, the per-iteration complexity is dominated by the computation of the gradient of the loss function. In general, it is difficult to give the running time of the gradient of the loss, as it depends on the choice of regularizer and constraints. But as an example, for NCA one must compute $G\boldsymbol{x}_{ij}\boldsymbol{x}_{ij}^T$ and $G\boldsymbol{x}_{ik}\boldsymbol{x}_{ik}^T$ for all $\boldsymbol{x}_{ij}$ and $\boldsymbol{x}_{ik}$; each such computation requires $O(rd)$ time. For projected gradient descent over A, often the gradient will require $O(d^2m)$ time, where d is the number of dimensions and m is the number of constraints, but again, the cost will depend on the particular formulation. The projection back onto the cone of positive semi-definite matrices requires $O(d^3)$ time. For Bregman projections and stochastic gradient descent, the running time per iteration (which now only looks at a single constraint) is typically $O(d^2)$ when fully optimized (for instance, by taking advantage of the fact that the projection onto the cone of positive semi-definite matrices can be performed efficiently after low-rank updates of A). Thus, iterating through all constraints typically yields $O(d^2m)$ total time.

In addition to the per-iteration cost, these optimization techniques also depend on the number of iterations to convergence. A full discussion of the convergence of these techniques is beyond the scope of this article, as it depends on various factors including choice of step size and properties of the regularizer.

2.5.6 Other Models and Considerations

Naturally, our treatment of Mahalanobis metric learning models and algorithms does not cover all of the techniques proposed in the literature, and we add a few remarks to the above discussion. Recently, there has been additional work on the case of *mixed-norm* regularizers, such as the $(2,1)$-norm: $\|A\|_{(2,1)} = \sum_j (\sum_i A_{ij}^2)^{1/2}$. In [82], a particular Mahalanobis metric learning problem with relative distance constraints was shown to be equivalent to a squared trace-norm formulation with relative distance constraints. Other authors have considered ℓ_1-regularization on the entries of A, e.g., in [55]. Another recent paper showed that several Mahalanobis metric learning methods can be shown to be equivalent to a problem of minimizing the maximal eigenvalue of a symmetric matrix [83]; this yields new algorithms for existing models, such as LMNN, as well as new formulations.

We also note that there has been some recent interest and work in proving generalization bounds for Mahalanobis metric learning methods, but these techniques are beyond the scope of this survey. We refer the reader to [9, 40] for further details.

2.6 Summary

In this section, we focused on a general model for linear metric learning. This is known as *Mahalanobis metric learning* in the literature, and accounts for most of the existing research in metric learning. Our goal was to present as unified a view as possible, such that a wide variety of existing work could be viewed as special cases of a general model. As such, we studied a regularized metric learning model which contains a regularizer on the linear transformation along with loss functions that encode the supervision.

The remainder of the section considered more deeply examples of the general linear transformation model that have been studied, along with a review of optimization techniques that have been employed on these models.

3

Nonlinear Models for Metric Learning

In the previous section, we considered a general framework for learning global linear metrics. In this section, we consider two alternate but related approaches for metric learning. First, we will consider global nonlinear methods, which learn distances of the form $\|f(\boldsymbol{x}) - f(\boldsymbol{y})\|_2$ for some function f; as before, we are mainly concerned with models and algorithms for learning such f under the presence of supervision over the learned distances or similarities. Second, we will explore local distance learning methods, both linear and nonlinear, which aim to learn several distance functions over the domain of the data.

Recall that a linear transformation may be viewed as "stretching" (or "shrinking") the axes and rotating the data. To motivate the need for nonlinear or non-global metrics, consider the data in Figure 3.1, an extension of the classic XOR example to demonstrate the limitations of linear methods. Here the data roughly can be clustered into four "blobs." Shrinking the x-axis in this figure, for instance, would yield a linear transformation that brings blue points closer to green points, and red points closer to black points while maintaining separation along the y-axis. The problem arises when we require that the red and green points be similar, the blue and black points be similar, but that the red

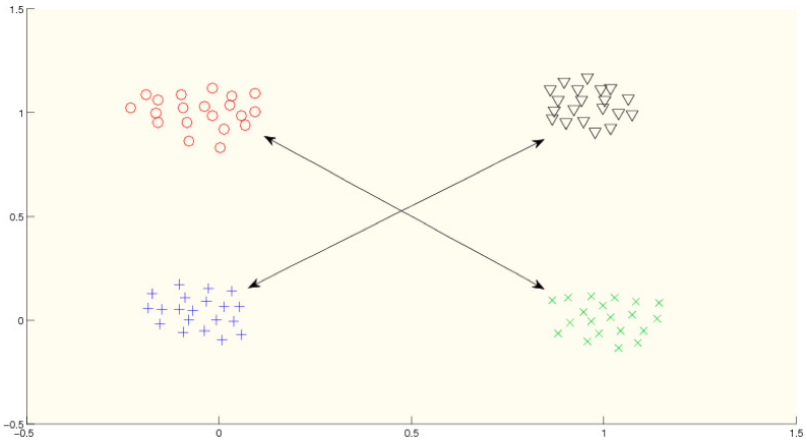

Fig. 3.1 Example of the limitation of linear methods. Suppose in this data we would like that the black and blue points should be "similar" to one another, while the red and green point should also be similar to one another (while simultaneously the black and blue points should be dissimilar to the red and green points). No global linear metric will suffice for enforcing the constraints.

and green are dissimilar to the blue and black points. Here there is no way to construct a linear transformation to satisfy such constraints.

In general, learning a nonlinear transformation is difficult — unlike linear transformations, which can be expressed as a matrix of parameters, the set of nonlinear transformations is not readily parametrized. In order to learn such transformations, it will be necessary to restrict the form of the nonlinear mapping f to a particular parameterized class, and in such a way that parameters can be learned efficiently. We will focus on one such parameterization that has been successful for metric learning, namely kernelized linear transformations.

3.1 Kernelization of Linear Methods

Recall the linear model discussed in the previous section:

$$\min_{A \succeq 0} r(A) + \lambda \sum_{i=1}^{m} c_i(X^T A X),$$

where r is the regularizer and c_i are the loss functions. The algorithms discussed for optimizing this model generally require updating A iteratively, using the data points from X directly. What we will show

in the following is that we can design algorithms that do not access the data directly; instead, they only require access to inner products $\boldsymbol{x}_i^T \boldsymbol{x}_j$ between data points. Because we only require such inner products, we can generalize the resulting algorithms to utilize *kernel functions* instead of inner products. Such kernel functions represent inner products in a high or infinite dimensional space, and applying such algorithms will correspond to learning nonlinear transformations in the input space. More specifically, a kernel function can be written as $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$ for some (generally nonlinear) function ϕ. When we apply linear metric learning algorithms with this inner product, it corresponds to learning a linear transformation in the space of the ϕ function (i.e., the feature space). Therefore, we can write these methods as learning distances of the form $\|G\phi(\boldsymbol{x}) - G\phi(\boldsymbol{y})\|_2$, which is a nonlinear transformation over the input data in general. We will see that the linear transformation model permits such kernelization for a large class of regularizers, called *spectral regularizers*, and that in most cases algorithms for the linear case can easily be generalized to algorithms for the nonlinear case.

3.1.1 Brief Review of Kernel Methods

We briefly review some background on kernel methods; for a more complete treatment, see [59, 64]. As stated above, a main use of kernel methods is in extending linear learning methods to the nonlinear case, and this is achieved by writing algorithms in terms of inner products and replacing those inner products with kernel functions. As a very simple example, consider a Euclidean nearest neighbor classifier. For each query $\boldsymbol{x}_q$, we compute the distance to each point $\boldsymbol{x}$ in the database via $\|\boldsymbol{x}_q - \boldsymbol{x}\|_2$, and then classify $\boldsymbol{x}_q$ based on the labels of the nearest neighbors. Using the fact that $\|\boldsymbol{x} - \boldsymbol{y}\|_2^2 = (\boldsymbol{x} - \boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{y})$, we can write

$$\|\boldsymbol{x}_q - \boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}_q^T \boldsymbol{x}_q - 2\boldsymbol{x}^T \boldsymbol{x}_q + \boldsymbol{x}^T \boldsymbol{x}}.$$

Replacing the inner products with a kernel function $\kappa(\boldsymbol{x}_q, \boldsymbol{x}) = \phi(\boldsymbol{x}_q)^T \phi(\boldsymbol{x})$, the distance function is expressed as

$$\sqrt{\kappa(\boldsymbol{x}_q, \boldsymbol{x}_q) - 2\kappa(\boldsymbol{x}_q, \boldsymbol{x}) + \kappa(\boldsymbol{x}, \boldsymbol{x})}.$$

The resulting distance is a "kernelized" distance between the query and a datapoint. As a slightly more complex example, consider extending the k-means algorithm to kernel space [60]. Recall that, in each step of k-means, we compute the squared Euclidean distance from each point $\boldsymbol{x}$ to each cluster centroid $\boldsymbol{\mu}_c$. Since the cluster centroid is simply the mean of the data points in cluster π_c, we can write the distance computation as:

$$\|\boldsymbol{x} - \boldsymbol{\mu}_c\|_2^2 = \boldsymbol{x}^T\boldsymbol{x} - \frac{2\sum_{\boldsymbol{x}_i \in \pi_c} \boldsymbol{x}^T\boldsymbol{x}_i}{|\pi_c|} + \frac{\sum_{\boldsymbol{x}_i, \boldsymbol{x}_j \in \pi_c} \boldsymbol{x}_i^T\boldsymbol{x}_j}{|\pi_c|^2},$$

which is readily extended to kernel space by replacing the above inner products with kernel functions.

In order for a kernel function to be valid, it must represent an inner product between data points in the Hilbert space induced by the mapping ϕ. One way to express this requirement (using Mercer's theorem) is that any matrix K of kernel function values defined over a set of data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ (so $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$) must always be positive semi-definite (equivalently, it must have non-negative eigenvalues). For instance, in the case of the standard inner product, the resulting K is simply the Gram matrix X^TX, and is positive semi-definite.

In practice, typically one chooses a kernel from a set of known kernel functions. Two popular choices are the polynomial kernel and the Gaussian kernel. The polynomial kernel is defined as $\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y} + c)^d$ for positive reals c and d. Consider the simple case of $c = 0$, $d = 2$, and data from 2 dimensions (so each $\boldsymbol{x} = [x_1; x_2]$). Then we can see that

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{x}^T\boldsymbol{y})^2 = \phi(\boldsymbol{x})^T\phi(\boldsymbol{y}),$$

where $\phi(\boldsymbol{x}) = [x_1^2; \sqrt{2}x_1x_2; x_2^2]$. The Gaussian kernel is defined as $\kappa(\boldsymbol{x}, \boldsymbol{y}) = \exp(-\|\boldsymbol{x} - \boldsymbol{y}\|_2^2/(2\sigma^2))$. The resulting embedding $\phi(\boldsymbol{x})$ can be shown to be infinite-dimensional in this case.

A further benefit to utilizing kernel methods lies in the fact that kernel functions have been designed and successfully employed for several types of structured data [24]. For instance, one can define a kernel between two strings, which can be viewed as embedding a string into a high-dimensional feature space and then computing inner products

efficiently in this feature space [50]. Another example is in computer vision, where it is common to use kernels defined between images. For example, the pyramid match kernel [29] defines a notion of similarity between images based on comparing sets of local features between the images; this kernel has the property that, while the underlying representation of the feature space is extremely high dimensional, the kernel itself can be computed cheaply. Given the prevalence of kernels in these applications, kernelization of linear metric learning will make it possible to apply metric learning to domains such as computer vision where feature spaces can be very high but where efficient kernels have been developed.

3.1.2 Spectral Regularizers

In order to extend global linear metric learning methods to kernel space, it will be necessary to consider a restricted class of regularizers known as *spectral functions*. Such functions limit kernelization to a smaller set of problems, but as we will see, in practice most global linear metric learning methods use spectral regularizers.

We define spectral functions as follows:

Definition 3.1. We say that $f : \mathbb{R}^{n\times n} \to \mathbb{R}$ is a **spectral function** if $f(A) = \sum_i f_s(\lambda_i)$, where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A and $f_s : \mathbb{R} \to \mathbb{R}$ is a real-valued scalar function. Note that if f_s is a convex scalar function, then f is also convex.

Example 3.1. Consider the function $r(A) = \mathrm{tr}(A)$. This is clearly a spectral function, as the matrix trace is the sum of the eigenvalues $\lambda_1, \ldots, \lambda_d$ of A, and so the resulting scalar function is $f_s(\lambda_i) = \lambda_i$. The trace function is convex, but not strictly convex. More generally, the regularizer $r(A) = \mathrm{tr}(AC)$ can be written as $r(A) = \sum_i w_i \lambda_i$, where $w_i = \boldsymbol{v}_i^T C \boldsymbol{v}_i$ and $\boldsymbol{v}_i$ is the ith eigenvector of A. This does not satisfy the definition of a spectral regularizer, but as we will see, our results can be extended for this function in some cases.

Example 3.2. Consider the squared Frobenius norm $r(A) = \|A\|_F^2$. Using the fact that $\|A\|_F^2 = \text{tr}(A^T A)$, then one can easily show using the eigendecomposition of A that $r(A) = \sum_i \lambda_i^2$. Therefore, $f_s(\lambda_i) = \lambda_i^2$ and the resulting function is a spectral function. This function is strictly convex.

Example 3.3. The rank function $r(A) = \text{rank}(A)$ is also a spectral function. The corresponding scalar function is $f_s(\lambda_i) = 1$ if $\lambda_i \neq 0$, and 0 otherwise. This is not a convex function.

Example 3.4. The function $r(A) = \text{tr}(A) - \log\det A$, as utilized by the ITML algorithm, is a spectral function. We saw earlier that trace is a spectral function, and $\log\det A = \sum_i \log(\lambda_i)$, so $r(A) = \sum_i (\lambda_i - \log\lambda_i)$. This is a strictly convex function.

Example 3.5. The function $\text{tr}(A\log A - A)$ is also a spectral function. The corresponding scalar function in this case is $f_s(\lambda_i) = \lambda_i \log\lambda_i - \lambda_i$, and is a strictly convex function.

Note that the examples above encompass many of the models discussed in the previous section. For instance, the squared Frobenius norm is used by POLA and the method of Schultz and Joachims.

3.1.3 A Representer Theorem

We will now show an important result, namely that the linear model for metric learning discussed in the previous section can be solved in kernel space when the regularizer is a spectral function. That is, we will re-write the optimization problem with respect to the kernel matrix $K = X^T X$, and once we have the kernel matrix we will not require the original matrix X. This permits the use of kernel functions to form kernel matrices after applying nonlinear mappings to the original data.

More specifically, we will consider the following problem:

$$\min_{K_A \succeq 0} r(K^{-1/2} K_A K^{-1/2}) + \lambda \sum_{i=1}^{m} c_i(K_A), \tag{3.1}$$

and we will show that this problem is equivalent to the linear metric learning problem, in that the solution to one problem can be obtained in closed form from the solution to the other (and vice versa).

Theorem 3.1. Let $K = X^T X$ and r be a strictly convex spectral function. Denote the global minimum of the corresponding scalar function r_s as $\alpha \geq 0$. Let A^* be an optimal solution to the linear model (2.1) and K_A^* be an optimal solution to the kernel problem (3.1). Then

$$A^* = \alpha I + XSX^T,$$

where $S = K^{-1}(K_A^* - \alpha K)K^{-1}$. Furthermore, $K_A^* = X^T A^* X$.

The proof is given in the appendix. A key consequence of the theorem is that A^* can be constructed in closed-form from K_A^* and vice versa, thus allowing us to solve whichever problem is more convenient. In the kernel setting, where the underlying feature space may be extremely high-dimensional (or even infinite-dimensional), then it is clear that solving (3.1) is typically easier than solving (2.1). Note that the above result is a representer theorem, in that it shows that the optimal solution to the linear metric learning problem can be expressed as an appropriate expansion in terms of the training data, as given by X. As such, the result is similar in style to the representer theorem for support vector machines [59].

Another key advantage of the above theorem is that it shows us, constructively, how to compute inner products or Mahalanobis distances in the learned space, even if we are working in kernel space. Suppose that we have learned a matrix A via optimization of (2.1). Then, given arbitrary points $\boldsymbol{x}$ and $\boldsymbol{y}$, computing the mapped inner product or Mahalanobis distance is easily computed as $\boldsymbol{x}^T A \boldsymbol{y}$ and $(\boldsymbol{x} - \boldsymbol{y})^T A (\boldsymbol{x} - \boldsymbol{y})$, respectively. When we instead solve (3.1), we obtain K_A, not A. The theorem tells us that we could construct A via $A = \alpha I + XSX^T$, but

this would be expensive when working in high-dimensional feature spaces. Instead, we can simply compute the mapped inner product $\boldsymbol{x}^T A\boldsymbol{y}$ using an *implicit* representation of A using K_A and kernel evaluations, as follows:

$$\boldsymbol{x}^T A\boldsymbol{y} = \boldsymbol{x}^T(\alpha I + XSX^T)\boldsymbol{y} = \alpha\kappa(\boldsymbol{x},\boldsymbol{y}) + \boldsymbol{k}_x^T S\boldsymbol{k}_y,$$

where $\boldsymbol{k}_x = [\kappa(\boldsymbol{x},\boldsymbol{x}_1),\ldots,\kappa(\boldsymbol{x},\boldsymbol{x}_n)]^T$ and $\boldsymbol{k}_y = [\kappa(\boldsymbol{y},\boldsymbol{x}_1),\ldots,\kappa(\boldsymbol{y},\boldsymbol{x}_n)]^T$, and S is obtained from K and K_A. This requires $2n+1$ kernel evaluations per mapped inner product computation; further, computing the term $\boldsymbol{k}_x^T S\boldsymbol{k}_y$ involves $O(n^2)$ computations. When computing mapped inner products from test data to training data, however, we can reduce the overall complexity of computing the mapped inner product between a test point and a training point to $O(n)$ by precomputing $S\boldsymbol{k}_y$ for each training point.

3.1.4 The Kernel PCA Trick

In the previous section, we saw that the Mahalanobis metric learning problem could be expressed equivalently in a form amenable to kernels, and further that the learned distances could be computed via kernel functions. An elegant corollary of this analysis, studied by Chatpatanasiri et al. [11], shows that in some cases this leads to a simple method for performing the kernelization, wherein we simply apply kernel PCA to the data and run the same algorithm for the linear model on the transformed data. This is guaranteed to find the optimal solution to the kernel form of the metric learning problem, and does not require any additional algorithmic development.

Here we will describe this approach in more detail. The simplest scenario considered in [11] looks at the case where there is no regularization; that is, the linear metric learning problem can be expressed as a minimization only over the constraints, which we will write as

$$\min_{A\succeq 0} \sum_i c_i(X^T AX) \equiv \min_{A\succeq 0} f(X^T AX).$$

Theorem 3.1 above says that A has a representation $A = XSX^T$. Thus we could equivalently write the problem as

$$\min_{S\succeq 0} f(X^T XSX^T X) = f(KSK).$$

Now, let us denote the SVD of X as $X = U\Sigma V^T$. The kernel matrix is expressed as $K = V\Sigma^2 V^T$, and we have

$$\begin{aligned} f(KSK) &= f(V\Sigma^2 V^T S V \Sigma^2 V^T) \\ &= f(V\Sigma U^T U \Sigma V^T S V \Sigma U^T U \Sigma V^T) \\ &= f(X^T U \Sigma V^T S V \Sigma U^T X) = f(X^T U A' U^T X), \end{aligned}$$

where $A' = \Sigma V^T S V \Sigma$. Thus we can equivalently write the optimization in terms of $A' : \min_{A' \succeq 0} f(X^T U A' U^T X)$. We also note that, if A is the optimal solution to the linear metric learning problem, then

$$\boldsymbol{x}_i^T A \boldsymbol{x}_j = \boldsymbol{x}_i^T X S X^T \boldsymbol{x}_j = \boldsymbol{x}_i^T U A' U^T \boldsymbol{x}_j.$$

Putting these two observations together, we can equivalently run the metric learning problem on $\tilde{X} = U^T X$ to produce a matrix A'. Then, the learned distances are computed via learned inner products $\tilde{\boldsymbol{x}}_i^T A' \tilde{\boldsymbol{x}}_j$.

However, the data $\tilde{X}$ is simply the original data projected onto the eigenvectors of the covariance matrix, which is precisely what is computed by PCA (modulo the centering step, which is not required here). As kernel PCA can perform this projection when the data has been mapped into kernel space, this yields a constructive and simple approach to performing metric learning in kernel space when the problem has no regularizer (or the regularizer is of the form $r(X^T A X)$):

- Given original data X, compute $\tilde{X}$ by projecting onto the eigenvectors of the covariance with non-zero eigenvalues (via the approach used by kernel PCA).
- Run an existing algorithm for Mahalanobis metric learning over f, using $\tilde{X}$ in place of X, to learn A'.
- Learned distances in kernel space are computed via $d_{A'}(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j)$.

Two examples discussed in [11] for utilizing the kernel PCA trick are NCA and LMNN. In both cases, we can write the entire optimization as in the above discussion, namely a minimization of $f(X^T A X)$. For instance, in LMNN, recall that the regularizer is written as $\mathrm{tr}(AC)$, where $C = \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{S}} (\boldsymbol{x}_i - \boldsymbol{x}_j)(\boldsymbol{x}_i - \boldsymbol{x}_j)^T$. It is straightforward to see that this regularizer is of the form $r(X^T A X)$ and can therefore be

incorporated into the above approach. Similarly, we saw that NCA uses no explicit regularizer.

The authors of [11] go on to prove an additional result, where they show that when a Hilbert–Schmidt regularizer is added, the kernel PCA approach can still be applied. However, for general spectral regularizers such as that used by ITML, one cannot simply apply kernel PCA to the data. Finally, we also note the related work of Torresani and Lee [71], an earlier paper that considers the use of kernel PCA for solving LMNN in kernel space. The authors demonstrate first how to perform gradient descent on LMNN on the low-dimensional matrix G (where $A = G^T G$), as in NCA. Then they show that the gradient descent update can actually be applied in kernel space in a way that is similar to [11]. Though the authors do not formally prove that the optimal solution of LMNN can be represented via the training data, this paper was an early step on the path to understanding kernelization of Mahalanobis metric learning.

3.1.5 Additional Kernelization Examples

To make the application of Theorem 3.1 clear, let us consider some additional kernelization examples. First, we consider ITML. Recall the problem in its constrained form:

$$\begin{aligned} \min_{A \succeq 0} \quad & \operatorname{tr}(A) - \log\det A \\ \text{such that } & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq u \ (i,j) \in \mathcal{S} \\ & d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq \ell \ (i,j) \in \mathcal{D} \end{aligned}$$

As discussed earlier, the regularizer is a spectral function, and the constraints are clearly a function of the matrix $X^T A X$, since the data is only utilized to compute Mahalanobis distances. We can therefore apply Theorem 3.1 directly to this problem, and we must simply determine the appropriate form of the objective and constraints for the kernel form of the optimization problem.

As given in the theorem, the regularizer in kernel space is of the form $r(K^{-1/2} K_A K^{-1/2})$; as we discussed in Section 2.4.3, this is equal to $D_{\ell d}(K_A, K)$, where $D_{\ell d}(\cdot, \cdot)$ is the LogDet divergence. In terms of

the constraints, we have

$$(\boldsymbol{x}_i - \boldsymbol{x}_j)^T A(\boldsymbol{x}_i - \boldsymbol{x}_j) = (\boldsymbol{e}_i - \boldsymbol{e}_j)^T K_A(\boldsymbol{e}_i - \boldsymbol{e}_j),$$

where $K_A = X^T AX$. Thus, the kernel form of ITML is the following kernel learning problem:

$$\begin{aligned} \min_{K_A \succeq 0} \quad & D_{\ell d}(K_A, K) \\ \text{such that } & K_A(i,i) + K_A(j,j) - 2K_A(i,j) \leq u \ (i,j) \in \mathcal{S} \\ & K_A(i,i) + K_A(j,j) - 2K_A(i,j) \geq \ell \ (i,j) \in \mathcal{D} \end{aligned}$$

Further, the optimal solution for A is of the form $A = I + XSX^T$ (here, $\alpha = 1$, which can be seen by minimizing the scalar function $r_s(\lambda) = \lambda - \log \lambda$ as required by the theorem) and S is computed as given in the theorem.

Thus, for ITML, the kernel problem seeks to find a kernel matrix K_A that is "close" to K (in terms of the LogDet divergence), but where squared Euclidean distance constraints using the learned kernel K_A are satisfied. Note that, as with standard ITML, we often incorporate slack variables into this formulation or solve the corresponding unconstrained version, as there may not be a feasible solution to the above problem. Moreover, we note that this kernel formulation of the problem was actually studied before the corresponding ITML formulation was proposed, but the connections to Mahalanobis metric learning were not known before the publication of ITML [45].

As another example, consider the following squared-Frobenius regularization problem, similar to the problem of POLA:

$$\begin{aligned} \min_{A \succeq 0} \quad & \|A\|_F^2 \\ \text{such that } & y_{ij}(b - d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)) \geq 1, \ (i,j) \in \mathcal{P}, \end{aligned}$$

where $y_{ij} = 1$ if i and j are similar and -1 if they are dissimilar, and $\mathcal{P}$ is the set of pairs with constraints. In this case, we have $r(A) = \frac{1}{2}\|A\|_F^2$, which is again a spectral regularizer; here, $\alpha = 0$ and the optimal A is of the form XSX^T. So, applying Theorem 3.1, we have that the objective in kernel space is $\frac{1}{2}\|K^{-1/2} K_A K^{-1/2}\|_F^2 = \frac{1}{2}\|K_A K^{-1}\|_F^2$. Writing the constraints in kernel space analogously to ITML (we write each of

the m constraints as $g_i(K_A) \leq b_i$) for simplicity), we obtain the following kernel problem:

$$\begin{aligned} \min_{K_A \succeq 0} \quad & \|K_A K^{-1}\|_F^2 \\ \text{such that } & g_i(K_A) \leq b_i, \quad 1 \leq i \leq m. \end{aligned}$$

Finally, consider the trace-norm regularizer $r(A) = \text{tr}(A)$. Again, this is a spectral regularizer whose corresponding α value equals 0. Though Theorem 3.1 applies to strictly convex functions, and the trace-norm is not strictly convex, the representer theorem can also be shown to apply to the trace-norm [36]. Thus, the optimal A is of the form XSX^T, as with the squared Frobenius norm. After applying the theorem, the resulting objective is $\text{tr}(K_A K^{-1})$. The trace-norm regularizer tends to find low-rank matrices A, and this property is inherited when we move to kernel space. This leads to a type of *nonlinear* dimensionality reduction: since $A = XSX^T$ is low-rank at optimality, then S is a low-rank matrix, leading to a new representation of the data given by $\boldsymbol{x} \rightarrow S^{1/2}X^T\boldsymbol{x}$, which will be of low-dimensionality for low-rank S.

3.1.6 Scalability in Kernel Space

One of the challenges when applying Mahalanobis metric learning in kernel space arises when the number of data points n becomes large. The kernelization result, as shown in Theorem 3.1, indicates that the number of parameters to optimize changes from $O(d^2)$ to $O(n^2)$: instead of learning a Mahalanobis matrix A, we instead learn a kernel matrix K_A over all of the training data.

Performing metric learning in kernel space when the data set size is very large remains an ongoing research problem. Some simple approaches have been employed, and empirically seem to work well. For instance, when applying kernel PCA (to LMNN, for example), instead of projecting using *all* eigenvectors and eigenvalues of the kernel matrix, one could project using only the top eigenvectors and eigenvalues. Alternately, one can perform kernel PCA using a sample of the data, compute the kernel matrix over this set of sampled data points, and then perform the kernel PCA projection based only on this matrix.

For the more general setting of Theorem 3.1, some work along similar lines has been proposed. Analogous to the kernel PCA-based approaches, the main idea is to consider a smaller sample of the data, and constrain the optimal A matrix to admit a representation only in terms of these sampled data points. We refer the reader to [35] for a more in-depth discussion of these ideas.

3.2 Other Nonlinear Methods

We conclude this section by discussing a few other prominent metric learning methods.

3.2.1 Non-Mahalanobis Local Distance Functions

A major drawback to global Mahalanobis metric learning is that one learns a single transformation to be applied across the entire space globally. Intuitively, one can imagine that such a single transformation may be insufficient in many cases. For example, we may want to have a different metric for each class or cluster in the data.

The method of Frome et al. [23] takes this idea even further: they propose to learn a distance per training point. Given training data of n data points, each of which has d features, let $\boldsymbol{d}_{ij}(m)$ be some baseline distance between the mth feature of training points i and j. In simple settings, where each feature is a scalar, this distance can simply be the squared difference between the feature values. (As considered in [23], however, each feature is a multi-dimensional vector obtained by a feature extractor on an image patch, and additionally a matching algorithm is applied to match features from pairs of images.) We then define the learned distance between data point i and data point j as $\boldsymbol{w}_i^T \boldsymbol{d}_{ij}$, where $\boldsymbol{w}_i$ is a vector of d weights for training point i. Thus, the idea is that each training image i will have associated with it a vector $\boldsymbol{w}_i$ whose entries weight the features differently. Note that this implies that distances are no longer symmetric.

The goal of learning is to find the weight matrix $W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_n]$ over the n training points such that a regularized loss function is minimized. In fact, the resulting optimization problem turns out to be quite similar to the formulations described earlier, with the only

exception being that the Mahalanobis distance is replaced by the local distances as described above. More specifically, the authors of [23] consider relative distance constraints: given a triple $(i, j, k) \in \mathcal{R}$ such that the distance between i and j should be smaller than the distance between i and k, we can set this up as a constraint of the form:

$$\boldsymbol{w}_i^T \boldsymbol{d}_{ik} > \boldsymbol{w}_i^T \boldsymbol{d}_{ij} + 1.$$

As with Mahalanobis distances, we can encode this into a loss function; the authors utilize the hinge-loss $[1 + \boldsymbol{w}_i^T \boldsymbol{d}_{ij} - \boldsymbol{w}_i^T \boldsymbol{d}_{ik}]_+$. The final step is to add a regularizer, and they choose a squared Frobenius norm on W (additionally, the entries of W are constrained to be non-negative). This leads to the following problem:

$$\min_{W \geq 0} \frac{1}{2}\|W\|_F^2 + \lambda \sum_{(i,j,k) \in \mathcal{R}} [1 + \boldsymbol{w}_i^T \boldsymbol{d}_{ij} - \boldsymbol{w}_i^T \boldsymbol{d}_{ik}]_+.$$

Presented in this manner, there are clear similarities between this formulation and the method of Schultz and Joachims, which performs diagonal Mahalanobis learning under a squared Frobenius regularizer and a hinge-loss over relative distance constraints. Indeed, we can stretch the connections further by showing how the Frome et al. approach could be viewed as the Schultz and Joachims method in the case where features are scalars and the baseline distance is simply the squared difference (we note that this connection does not appear to be particularly insightful, and we discuss it simply for the purposes of presenting as unified a framework for distance learning as possible).

Suppose for simplicity that we have 3 training data points. Let us represent $W = [\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3]$ alternatively as

$$\tilde{W} = \begin{bmatrix} \mathrm{diag}(\boldsymbol{w}_1) & 0 & 0 \\ 0 & \mathrm{diag}(\boldsymbol{w}_2) & 0 \\ 0 & 0 & \mathrm{diag}(\boldsymbol{w}_3) \end{bmatrix}.$$

Now consider the Mahalanobis distance (using $\tilde{W}$) between $[\mathbf{0}; \boldsymbol{x}_1; \mathbf{0}]$ and $[\mathbf{0}; \boldsymbol{x}_2; \mathbf{0}]$; this is precisely the weighted local distance using the weights from training point 2. Thus, conceptually, we can imagine that we have n^2 training data points — for each data point $\boldsymbol{x}_i$, we encode

it n ways into a nd-dimensional space, where we change the placement of $\boldsymbol{x}_i$ for each. Utilizing these n^2 data points, we can then set up a Mahalanobis distance learning problem in the manner of Schultz and Joachims, which is easily seen to be equivalent to the learning problem of Frome et al. At test time, when computing the distance between a training point i and some novel test point, we utilize the representation of the test point which is all zeros except in the i-th "slot," which contains the data point.

3.2.2 Mahalanobis Local Distance Functions

Weinberger and Saul considered an alternative local distance learning strategy, multiple metric LMNN (MM-LMNN) [77]. The idea is to learn a Mahalanobis matrix per class. Recall that LMNN utilizes a linear objective with relative distance constraints. The only difference between standard LMNN and MM-LMNN is that, whenever one computes the Mahalanobis distance between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, instead of using the global Mahalanobis distance one uses the Mahalanobis distance corresponding to the class of $\boldsymbol{x}_i$. That is, if ℓ_i corresponds to the class label of data point $\boldsymbol{x}_i$ (so $\ell_i \in \{1, \ldots, k\}$, where k is the total number of classes in the data), then we replace $d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)$ with $d_{A_{\ell_i}}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, and we learn matrices $A_1, \ldots, A_k$.

We note that, in principle, such an approach could be applied to any Mahalanobis distance learning algorithm, allowing one to extend a single global Mahalanobis matrix learning problem into a multiple Mahalanobis matrix learning problem.

We also mention the related approach of multi-task metric learning [54]. Briefly, this approach also learns multiple metrics, one per group of data; however, unlike the MM-LMNN formulation, we introduce a matrix A_0 into the formulation such that each Mahalanobis distance is of the form

$$d_t(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^T (A_0 + A_t)(\boldsymbol{x}_i - \boldsymbol{x}_j),$$

where t indexes one of the groups of data. Thus A_0 is shared across all of the Mahalanobis distances, and an additional regularizer is placed on A_0. See [54] for further details.

3.2.3 Metric Learning with Neural Networks

We next discuss two existing approaches to learning nonlinear metrics based on neural networks. Recall that for Mahalanobis distances we are learning a distance of the form $\|G\boldsymbol{x}_i - G\boldsymbol{x}_j\|_2$, where $A = G^T G$. When working in kernel space, we generalize this to $\|G\phi(\boldsymbol{x}_i) - G\phi(\boldsymbol{x}_j)\|_2$, but a more general nonlinear metric learning approach would consider alternative parameterizations for learning a distance of the form $\|\psi(\boldsymbol{x}_i) - \psi(\boldsymbol{x}_j)\|_2$.

The approach of Chopra et al. [13] considers learning a convolutional network in order to parameterize the mapping ψ. More specifically, given two input data points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we pass both through identical convolutional networks with common parameters. These networks are multi-layer, nonlinear systems, and produce a mapping from the input space to some (nonlinear transformed space). Though the details of convolutional networks are beyond the scope of this survey, we refer the reader to [13] for details on how such networks are set up and trained.

The underlying learning process utilizes a loss function which is somewhat reminiscent of the losses used by some of the techniques specified above. In particular, the approach considers a form of relative distance constraints (in the terminology of [13], if i and j should be more similar than i and k, they say i and j are a *genuine* pair and i and k an *impostor* pair). Denoting E_{ij} as the learned squared distance between a genuine pair and E_{ik} as the distance between an impostor pair, the loss function aims to minimize

$$\sum_{(i,j,k)\in\mathcal{R}} \alpha_1 E_{ij} + \alpha_2 \exp(-\alpha_3 \sqrt{E_{ik}}),$$

for appropriately chosen $\alpha_1, \alpha_2, \alpha_3$. While the underlying learning process and parameterization are quite different from the Mahalanobis learning approach, the loss function is similar.

A related technique was explored by Salakhutdinov and Hinton [58], who focused on extending NCA to the nonlinear case. Recall that the NCA probabilities can be expressed as:

$$p^A(j|i) = \frac{\exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_j))}{\sum_{k\neq i} \exp(-d_A(\boldsymbol{x}_i, \boldsymbol{x}_k))}.$$

Salakhutdinov and Hinton replace the Mahalanobis distance with a nonlinear distance function given by the distance between the outputs of a multi-layer neural network, in a style similar to that of Chopra et al. Thus, as in Chopra et al. the distances are parameterized by the weights of the neural network, and training seeks to learn these weights.

3.2.4 Other Nonlinear Extensions of LMNN

We conclude this section by overviewing two additional nonlinear extensions of LMNN, namely χ^2-LMNN and GB-LMNN [42].

Let us focus on the χ^2-LMNN variant. Recall that the LMNN problem seeks to minimize a linear term (a sum of learned distances over similar pairs) along with a set of relative distance constraints (encoded via a hinge loss):

$$\min_{A \succeq 0} \sum_{(i,j) \in \mathcal{S}} d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) + \lambda \sum_{(i,j,k) \in \mathcal{R}} [1 + d_A(\boldsymbol{x}_i, \boldsymbol{x}_j) - d_A(\boldsymbol{x}_i, \boldsymbol{x}_k)]_+.$$

Simply stated, the goal in χ^2-LMNN is to optimize the same objective, but with the χ^2 histogram distance in place of the Mahalanobis distance. In this manner, there are strong similarities to the approach of Frome et al., as discussed earlier, where the Mahalanobis distances are replaced with local distances to form a nonlinear method.

The χ^2-distance is defined as follows. Assuming two d-dimensional vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ defined over the d-dimensional simplex, we can define the distance as

$$\chi^2(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{2} \sum_{i=1}^{d} \frac{(\boldsymbol{x}(i) - \boldsymbol{y}(i))^2}{\boldsymbol{x}(i) + \boldsymbol{y}(i)}.$$

This distance has been used in a variety of domains, e.g., [16, 73, 75]. The authors of [42] propose a "Mahalanobis-like" extension of the χ^2-distance — instead of the standard distance, consider the following transformed distance:

$$\chi^2_G(\boldsymbol{x}, \boldsymbol{y}) = \chi^2(G\boldsymbol{x}, G\boldsymbol{y}),$$

where G is a linear transformation that maintains the data on the simplex. The authors discuss that the margin can no longer be simply

set to 1 as in standard LMNN, and must be optimized as an additional hyperparameter. Finally, the authors demonstrate how to perform optimization using a simple unconstrained sub-gradient descent method. See [42] for details of the method, as well as details of their other proposed extension of LMNN based on gradient boosting.

4

Extensions

In the basic metric learning model, we are concerned with learning an appropriate distance function tuned to some given task. As discussed previously, the end goal is typically not the metric itself, but rather some other task for which the learned metric will ideally improve performance. Nearest neighbor search is the most common, as many of the metric learning approaches are treated as optimization problems which attempt to shrink the distances between similar pairs of objects while expanding distances between dissimilar pairs.

However, the simple approach of applying metric learning for a nearest neighbor classification problem is not always the whole story. In this section, we discuss extensions of the basic metric learning approach to solve problems beyond a simple nearest-neighbor classifier.

4.1 Metric Learning for Kernel Regression

In some cases, metric learning can be used within other supervised learning methods. Consider the problem of regression. Suppose we have data points $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$, where the $\boldsymbol{x}_i$ are the data points and

the y_i are the (real-valued) response variables. In kernel regression, one approximates the y_i via

$$\hat{y}_i = \frac{\sum_{j \neq i} y_j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sum_{j \neq i} \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)}.$$

In [79], the problem of kernel regression was considered in a metric learning context. Often one uses a standard Gaussian kernel $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/(2\sigma^2))$. The approach in [79] replaces the squared Euclidean distance within the Gaussian kernel with a Mahalanobis distance utilizing $A = G^T G$. Then, gradient descent is performed on the simple squared loss $\sum_i (y_i - \hat{y}_i)^2$ with respect to G. One can imagine designing other learning problems involving Mahalanobis distances within a Gaussian kernel.

4.2 Metric Learning for Ranking

As an extension beyond the simple nearest-neighbor applications of most metric learning techniques, McFee and Lanckriet [52] present a method for utilizing Mahalanobis metric learning for the problem of ranking. In information retrieval settings, we often want to rank a set of data objects with respect to some query $\boldsymbol{q}$. One straightforward way to rank items would be to compute the distance between the query and each data item, and then rank by increasing distance.

Going a step further, we could attempt to learn a distance function such that the rankings obtained using these sorted distances optimize various known ranking measures such as the AUC, precision-at-k, MRR, MAP, or NDCG. McFee and Lanckriet consider learning a Mahalanobis metric in this framework. Here, the problem becomes one of structured prediction — one encodes constraints with respect to all permutations (rankings) over the data, leading to an enormous number of constraints, and each constraint uses some fixed ranking measure to compare pairs of rankings. As is common with structured prediction problems, enforcing all constraints using a standard optimization procedure is typically infeasible in practice, but cutting plane methods can be used to discover a small set of active constraints which are sufficient for optimization within some tolerance.

4.3 Dimensionality Reduction and Data Visualization

In the Mahalanobis distance learning framework, we can view metric learning algorithms as learning a linear transformation G to be applied to the data ($A = G^T G$), and where the resulting learned distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is of the form $\|G\boldsymbol{x} - G\boldsymbol{y}\|_2^2$. Typically, G is $d \times d$ (and full-rank), meaning that the transformation maintains the dimensionality of the input data. However, if the matrix A is not full-rank, then G will map the data from d dimensions to a lower-dimensional space, thereby performing dimensionality reduction. If dimensionality reduction is the goal (for example, for data visualization), then learning a low-rank A provides a possible way to perform a form of supervised dimensionality reduction. The main issue is in how to learn such a low-rank A, as most Mahalanobis metric learning methods tend to learn a full-rank A.

One approach, utilized by the authors of neighbourhood components analysis [26], is simply to perform gradient descent using G, where G is restricted to be $r \times d$ for some fixed r. The resulting problem is non-convex in G, but does guarantee that the resulting transformation will reduce the dimensionality to r. Further, this method, while discussed for NCA, could easily be applied to any metric learning formulation by replacing the optimization over A with a (typically non-convex) optimization over the $r \times d$ matrix G.

Another approach, introduced in [36], is to encourage low-rank A matrices via a trace-norm regularizer. As discussed earlier, this approach has the advantage of retaining convexity, but the resulting dimensionality of the mapped data r can be difficult to control directly — typically the trade-off parameter λ must be carefully selected to obtain a desired dimensionality. However, a benefit of trace-norm regularization is that the trace function is a spectral function, meaning that kernelization can be easily obtained. This leads to a form of *nonlinear* dimensionality reduction, as the mapping $G\phi(\boldsymbol{x})$ can be low-dimensional, even if the underlying feature space is very high (or even infinite-dimensional).

As an example of such nonlinear dimensionality reduction, Figure 4.1 shows a visualization of the MNIST digits, color-coded by

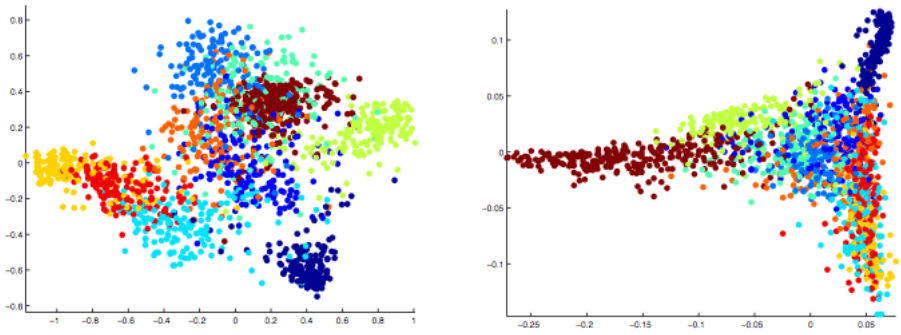

Fig. 4.1 Example taken from [36] demonstrating supervised nonlinear dimensionality reduction using kernelized Mahalanobis metric learning. The right plot shows the results of projecting to two dimensions via kernel PCA on the MNIST digits, while the left plot shows results when projecting using a learned rank-2 projection obtained by a trace-norm regularized kernelized metric learning problem.

their digit. On the right, we see the result of kernel PCA to perform nonlinear dimensionality reduction; in this case, there is no supervision, and the results are qualitatively poor. On the left, the visualization shows what a kernelized trace-norm metric learning optimization produces. Note that the digits shown here are not the same as the digits used for training (i.e., the visualized data utilizes the inductive representation for performing the mapping to the two dimensions). See [36] for further details.

Another example is shown in Figure 4.2, which is taken from [58]. This shows results comparing dimensionality reduction on MNIST, but where the comparison is between standard NCA (on the right) and its nonlinear extension via neural networks (on the left). In this case, we see clear evidence that a nonlinear metric learning method yields qualitatively better results than its linear counterpart.

4.4 Database Indexing

When the data set size is modest, metric learning can provide a useful preprocessing step for nearest neighbor searches. However, for very large-scale data, performing nearest-neighbor searches exhaustively can be prohibitive — given n total data points, the running time of a nearest-neighbor search grows linearly with n.

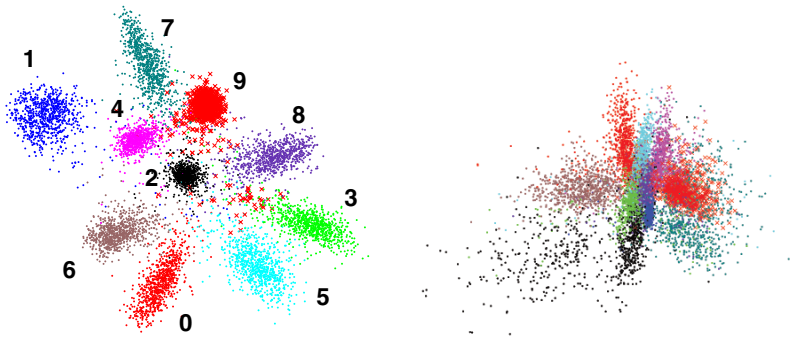

Fig. 4.2 Example reproduced from [58] showing further dimensionality reduction results on MNIST, comparing standard NCA (right) with its nonlinear extension based on neural networks (left), discussed in Section 3.2.3.

To deal with large-scale data sets, several approximate nearest neighbor search routines have been proposed, most notably locality-sensitive hashing [34] and search trees (e.g., [14, 22, 74]). Some recent work has explored how to most effectively combine metric learning with approximate nearest neighbor search data structures [38, 43]. For example, one standard locality-sensitive hashing function computes binary values of the form $\text{sign}(\boldsymbol{r}^T\boldsymbol{x})$, where $\boldsymbol{r}$ is a random vector drawn from the canonical Gaussian $\mathcal{N}(0, I)$. When performing metric learning, one additionally maps via G, leading to functions of the form $\text{sign}(\boldsymbol{r}^T G\boldsymbol{x})$. The work in [38, 43] explored how to compute these functions when running metric learning in kernel space, for particular kernels, resulting in a method that alternates between updating hash functions and updating the learned metrics. Another recent approach showed how to incorporate metric learning with ball trees effectively [77].

4.5 Domain Adaptation

In most metric learning applications, we think of a single data set $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$, and we consider mapped inner products of the form $\boldsymbol{x}_i^T A \boldsymbol{x}_j$. In a slight twist on this problem, consider two sources of data $X = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ and $Y = [\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m]$, along with inner products of the form $\boldsymbol{x}_i^T A \boldsymbol{y}_j$. In the latter problem, we need not require positive semi-definiteness of A.

To motivate this problem, consider two sources of image data. The data points (images) from X may consist of images taken with a high-resolution camera, while the data points in Y may consist of images taken with a low-resolution camera (e.g., a webcam). This is an example of *domain shift*, which has been extensively studied in the context of text [18] and images [44, 57]. Results have shown, for instance, that training on one domain and then testing on another domain often results in poor performance (see, e.g., [57]); it is therefore unlikely that a standard inner product $\boldsymbol{x}_i^T \boldsymbol{y}_j$ should be the most desirable notion of similarity between pairs of cross-domain data points.

One approach to the domain adaptation problem may be viewed as a generalization of the metric learning problem [44]. The idea is to learn a transformation A that maps the data from one domain to the other, thus leading to the inner product $\boldsymbol{x}_i^T A \boldsymbol{y}_j$. Note that such an approach can be applied even when the dimensionalities of the two domains are different (in which case the A matrix will not be square). Analogous to the standard linear metric learning problem, we can set up a regularized learning problem of the form:

$$\min_A r(A) + \lambda \sum_i c_i(X^T A Y).$$

Thus, the main differences between the above problem and a standard regularized linear metric learning problem are that the matrix is not constrained to be positive semi-definite and the constraints utilize the mapped inner products of the form $\boldsymbol{x}_i^T A \boldsymbol{y}_j$. Note that there are extensions of this model to the kernel case, analogous to the results of metric learning. See [44] for further details. We also refer the reader to earlier work [12], which also considers learning a non-PSD matrix A. In this case, the goal is not domain adaptation, but rather to speed up metric learning by avoiding projections onto the PSD cone; however, the resulting formulations are very similar to those of [44].

5

Applications

We conclude with a brief overview of some of the applications of metric learning employed in various domains. Our discussion of applications is far from being an exhaustive list, and is intended to convey some of the breadth of application domains considered recently for metric learning.

5.1 Computer Vision

Computer vision has seen many applications of metric learning, and is arguably the most successful domain for metric learning. Indeed, a large fraction of research in metric learning has been published in the computer vision community. In this section, we describe a few such efforts.

Before proceeding, we note that because it is beyond the scope of this survey, and since techniques vary from problem to problem, we will not describe the preprocessing methods utilized for encoding images (or image patches) into an appropriate Euclidean or reproducing kernel Hilbert space. We refer the reader to the specific papers for further details on such image preprocessing.

5.1.1 Image Retrieval and Classification

As we have discussed throughout this survey, a common end goal for metric learning is to improve nearest neighbor searches; after all, most metric learning formulations utilize constraints which aim to bring similar items closer together and dissimilar items farther apart. It is therefore natural to consider the problem of nearest-neighbor image retrieval as a potential application. Simply stated, the goal is to retrieve "similar" images to a query; metric learning is then used to refine an appropriate notion of similarity to aid in this task.

Many vision applications of metric learning, either explicitly or implicitly, perform image retrieval. An explicit example is from [43] (discussed earlier in Section 4.4), where the goal was to combine ITML metric learning with locality-sensitive hashing to simultaneously learn a good metric, as well as construct the appropriate hash functions for performing fast nearest neighbors searches at test time. As an example of this approach applied to retrieval and indexing, the authors considered performing metric learning over a database of 300,000 image patches taken from the Photo Tourism data set [33, 66]. This data consists of local image patches from several images (taken by different photographers) of the same underlying landmark. The training data consists of labels for matching patches across different photos, and the end goal is to retrieve matching image patches from other images to a query image patch.

As another example, many metric learning algorithms showed considerable success on the Caltech-101 data set.[1] Here, the problem is object classification: given a novel image, can we label it using an existing database of labeled images? The metric learning approach involves learning an appropriate metric, and then applying a simple nearest neighbor classifier to label the test images. The local metric approach of [23] produced state-of-the-art results at the time, and led to further work in the vision community on applications of metric learning.

Finally, we note that [32] provides another example of applying metric learning to the problem of image retrieval.

[1] Available at `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`

5.1.2 Face Recognition

The problem of face recognition seeks to determine whether two images of faces depict the same person or not. A recent paper [30] considered an approach to face recognition based on Mahalanobis distances.

Algorithmically, the main idea of the approach in [30] is to use a logistic regression-type loss within a Mahalanobis distance framework. Briefly, we have a set of n pairs of training points along with values $t_n \in \{0, 1\}$ that encode whether the pair of training images is a match or not. Analogous to logistic regression, we encode the probability that the pair of images $\boldsymbol{x}_i, \boldsymbol{x}_j$ are a match via

$$p_n = p(y_i = y_j \,|\, \boldsymbol{x}_i, \boldsymbol{x}_j, A, b) = \sigma(b - d_A(\boldsymbol{x}_i, \boldsymbol{x}_j)),$$

where y_i is a label for the face corresponding to image $\boldsymbol{x}_i$, b is a bias parameter, and $\sigma(\cdot)$ is the sigmoid function. The standard likelihood for $\boldsymbol{t} = (t_1, \ldots, t_n)$ is given by

$$p(\boldsymbol{t} \,|\, X, A, b) = \prod_n p_n^{t_n} (1 - p_n)^{1 - t_n}.$$

The corresponding log-likelihood is

$$\sum_n t_n \ln p_n + (1 - t_n) \ln(1 - p_n),$$

and can be viewed within the framework presented in Section 2 (where each n corresponds to a single constraint). The authors opt for a gradient descent approach to minimizing the above log-likelihood, and though they mention projected gradient descent as a natural way to enforce positive definiteness, they ultimately choose to ignore the positive definiteness constraint for scalability purposes.

On the Labeled Faces in the Wild (LFW) data set,[2] which contains 13,233 face images labeled by identity, the authors demonstrate state-of-the-art performance. The authors also demonstrate applications to face clustering and multi-class face recognition from a single face exemplar.

[2] Available at `http://vis-www.cs.umass.edu/lfw/`

5.1.3 Human Activity Recognition and Pose Estimation

A recent paper [72] considered the application of LMNN to the problem of human activity recognition in video sequences, and showed superior results with the learned metrics as compared to a number of other baselines. Here the problem is to determine what activity is being done in a video — for instance, activities could include people walking, running, or jumping. Using appropriate features, one can obtain very good performance on standard data sets via LMNN. Further, the authors demonstrated an application to video activity labelling on a set of YouTube videos.

In a related context, ITML was applied to the problem of human body pose estimation in [43]. The problem here is to predict the joint angles of a test image of a human body. This was set up as a metric learning problem as follows: a training set was generated using the software Poser [17], which generated 500,000 synthetic images of humans in various poses (with a variety of clothing, etc). Because the images were generated synthetically, the ground truth (joint positions and angles) of the underlying images is known.

Given this training data, a metric learning problem is set up where two training images should have distance close to their ground truth distance, which is simply the sum of the distances of corresponding joints angles of the two people in the images. After training, the joint angles of a novel test image is determined by computing the nearest neighbor from a test image to the training data, and using the joint angles of the nearest neighbors to predict the angles of the test image.

Some examples are given in Figure 5.1 of the test images and their nearest neighbors from the training set. Note that a similar approach was also applied in [62], where a method called parameter-sensitive hashing was applied to improve the nearest neighbor searches.

5.2 Text Analysis

A standard model for text documents is the TF-IDF model, along with the cosine similarity for computing a corresponding similarity score. Such an approach is known to achieve high precision for text retrieval

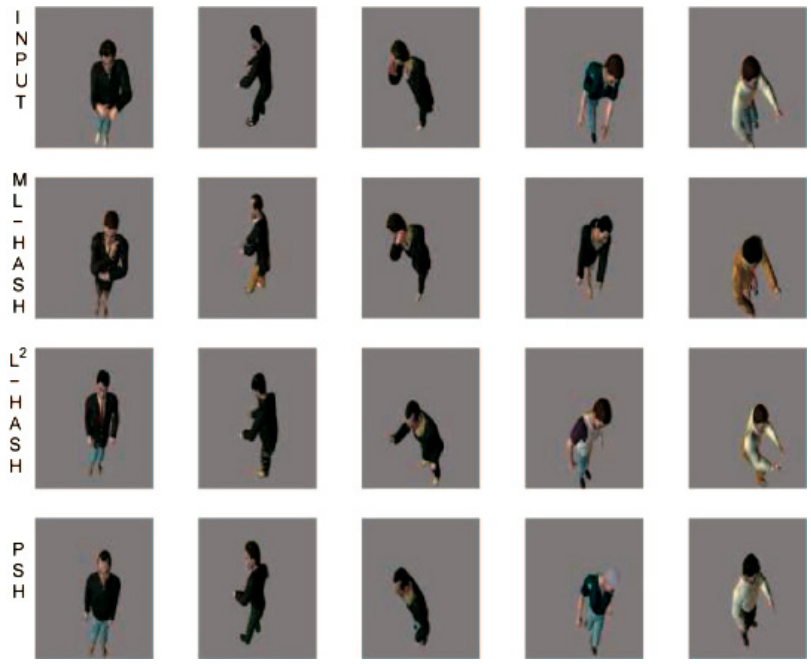

Fig. 5.1 Some example query poses along with the associated nearest neighbor as given by several retrieval methods (figure from [43]). ML-HASH refers to running metric learning and then applying LSH on top of the learned metrics. L2-HASH refers to using a baseline Euclidean distance with LSH. PSH refers to parameter-sensitive hashing [62].

applications, but recently in [19] it was shown that metric learning could achieve gains in terms of recall over standard similarities.

One of the challenges in applying metric learning to text is the high-dimensional nature of text — typically, one represents text data as sparse vectors whose dimensionality is equal to the number of unique words in the corpus (possibly after appropriate pruning). Given the sparsity of text, however, it turns out that kernelization is often natural in this setting, as the inner products between sparse text vectors can be easily computed. High-dimensional (kernelized) metric learning was explored in [19], where the kernel form of ITML was employed for text retrieval applications. As seen in Figure 5.2, the resulting metrics outperform standard approaches as measured by precision and recall.

We note that another approach to metric learning for text documents was explored in [48], although the approach from this paper

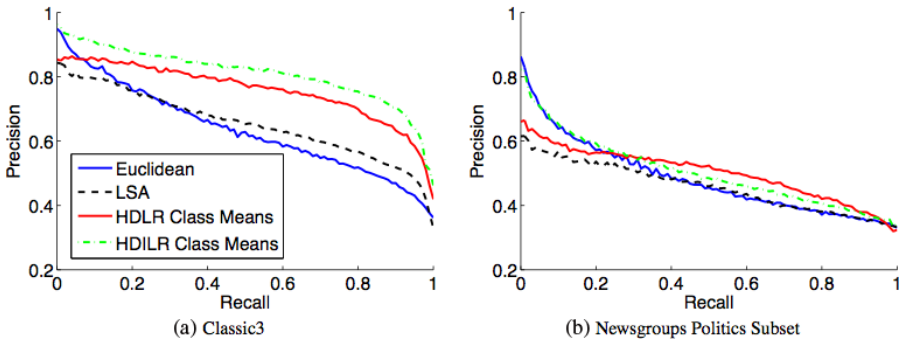

Fig. 5.2 Figure from [19] comparing Euclidean distance and Latent Semantic Analysis for comparing text documents with a metric learning approach. Here the red and green curves (HDLR Class Means and HDILR Class Means) refer to two high-dimensional metric learning algorithms. The precision-recall curves indicate that metric learning improves over the standard approaches. See the corresponding paper for further details.

does not fall into the same framework that has been discussed in this survey.

5.3 Other Applications

5.3.1 Music Analysis

In [65], Slaney et al. considered metric learning for music. The goal was to learn an appropriate notion of similarity between songs. The authors considered various approaches of how to do the embedding of the audio into an appropriate Euclidean space, and then considered several different existing metric learning algorithms, including NCA, LMNN, and RCA, for the metric learning step. According to their findings, NCA and RCA appear to perform the most robustly with the audio embeddings considered.

5.3.2 Automated Program Debugging

The original ITML paper briefly considered applying metric learning to the problem of discovering a distance between program executions. The authors utilized the output of a program called Clarify [31], which embeds program executions into a high-dimensional Euclidean space based on which functions in the program are executed, and other

statistics. Given a program execution that crashes due to a particular bug, the goal is to identify other existing executions from a database of executions that have the same bug. This is set up as a standard metric learning problem, and we refer the interested reader to [20] for further details.

5.3.3 Microarray Data Analysis

Two recent papers [67, 81] considered metric learning in the context of microarray data classification. DNA microarrays measure the expression levels of tens of thousands of genes simultaneously; a fundamental task is to classify tissue samples according to their gene expression levels. These predictions can then be used to help diagnose and predict various genetic disorders including cancer. While simple methods such as k-nearest neighbors have been used, metric learning has recently been shown to produce superior results in some cases.

6

Conclusions

In this survey, we attempted to provide an overview of recent work in metric learning, with a focus on models, extensions, and applications. One of the primary goals of this survey has been to present metric learning techniques in as common a framework as possible. For many techniques — particularly those that fall under the linear (Mahalanobis) metric learning model — this is straightforward, but there are several "outliers" that fall under metric learning but cannot be easily represented within the standard framework. We hope that the presentation has helped to unify the extensive literature to some degree. Naturally, this survey cannot give justice to all the literature on metric learning, and we have been forced to pick a subset of methods and ideas on which to focus.

Metric learning promises to continue to be a rich area of research. While linear methods appear to be fairly well-studied and understood, nonlinear methods continue to attract new research and ideas. In particular, scaling nonlinear methods to large data sets (e.g., when applying linear methods in kernel space) remains a difficult problem, and developing stronger theoretical underpinnings for methods involving local distances remains open. Further, given the recent progress in deep

learning methods, we expect that there will continue to be new nonlinear metric learning methods developed based on neural networks.

Applications of metric learning continue to emerge, and we expect this to continue moving forward. While metric learning has been used heavily for computer vision tasks, applications continue to emerge in biology, music, and multimedia. Ideally, such applications will help drive further theoretical and algorithmic development for metric learning.

A

Representer Theorem Proof

Here we prove Theorem 3.1, which allows one to develop nonlinear metric learning methods based on kernel methods. The proofs below follow the treatment in [35], but we include them here for completeness.

First, recall the linear transformation problem (2.1)

$$\min_{A \succeq 0} r(A) + \lambda \sum_{i=1}^{m} c_i(X^T A X),$$

and the corresponding kernel form of the problem (3.1)

$$\min_{K_A \succeq 0} r(K^{-1/2} K_A K^{-1/2}) + \lambda \sum_{i=1}^{m} c_i(K_A).$$

We also recall the representer theorem to be proved:

Theorem A.1(3.1). Let $K = X^T X$ and r be a strictly convex spectral function. Denote the global minimum of the corresponding scalar function r_s as $\alpha \geq 0$. Let A^* be an optimal solution to the linear model (2.1) and K_A^* be an optimal solution to the kernel problem (3.1). Then

$$A^* = \alpha I + X S X^T,$$

where $S = K^{-1}(K_A^* - \alpha K)K^{-1}$. Furthermore, $K_A^* = X^T A^* X$.

To prove the theorem, we first consider the following auxiliary problem:

$$\min_{A\succeq 0, L} f(A) + \lambda \sum_{i=1}^{m} c_i(X^T A X) \quad \text{s. t. } A = \alpha I + ULU^T, \tag{A.1}$$

where $L \in \mathcal{R}^{k\times k}$, $U \in \mathcal{R}^{d\times k}$ is a column orthogonal matrix, and I is the $d \times d$ identity matrix. Now we show the following:

Theorem A.2. Let f be a spectral function and let $\alpha \geq 0$ be any scalar. Then (A.1) is equivalent to

$$\min_{L\succeq -\alpha I} f(\alpha I + L) + \lambda \sum_{i=1}^{m} c_i(\alpha X^T X + X^T U L U^T X). \tag{A.2}$$

Proof. The constraint $A = \alpha I + ULU^T$ implies that there is a one-to-one mapping between A and L. We can therefore eliminate A from (A.1) by substituting $\alpha I + ULU^T$ for A. This results in

$$\min_{L\succeq -\alpha I} f(\alpha I + ULU^T) + \lambda \sum_{i=1}^{m} c_i(\alpha X^T X + X^T U L U^T X).$$

We also show that the objectives $f(\alpha I + L)$ and $f(\alpha I + ULU^T)$ are equal up to a constant. Let $U' \in \mathcal{R}^{d\times d}$ be an orthonormal matrix obtained by completing the basis represented by U, i.e., $U' = [U\ U_\perp]$ for some $U_\perp \in \mathcal{R}^{d\times(d-k)}$ such that $U^T U_\perp = 0$ and $U_\perp^T U_\perp = I$. Note that

$$A = \alpha I + ULU^T = U' \left(\alpha I + \begin{bmatrix} L & 0 \\ 0 & 0 \end{bmatrix} \right) U'^T.$$

For a spectral function f, we have that $f(VAV^T) = f(A)$, whenever V is an orthogonal matrix. Also,

$$f\left(\begin{bmatrix} B & 0 \\ 0 & C \end{bmatrix} \right) = f(B) + f(C).$$

Putting these observations together, we have

$$f(A) = f(\alpha I + ULU^T) = f\left(\begin{bmatrix} \alpha I + L & 0 \\ 0 & \alpha I \end{bmatrix}\right)$$
$$= f(\alpha I + L) + (d-k)f(\alpha),$$

establishing equivalence of the objectives up to a constant. □

We also establish the following result.

Theorem A.3. Suppose f, K, and α satisfy the conditions of Theorem 3.1. Then, the optimal solution to the linear model (2.1) is of the form $A^* = \alpha I + XSX^T$, where S is an $n \times n$ matrix.

Proof. Let $A = U\Lambda U^T = \sum_j \lambda_j \boldsymbol{u}_j \boldsymbol{u}_j^T$ be the eigenvalue decomposition of A. Consider the loss $c_i(X^T AX)$. If the jth eigenvector $\boldsymbol{u}_j$ of A is orthogonal to the range space of X (i.e., $X^T \boldsymbol{u}_j = 0$), then the eigenvalue λ_j is unconstrained except for the non-negative constraint imposed by positive semi-definiteness. Since the range space of X is at most n-dimensional, let us assume that $\lambda_j \geq 0, \forall j > n$ are not constrained.

Since f satisfies the conditions of Theorem 3.1, $f(A) = \sum_j f_s(\lambda_j)$. Also, $f_s(\alpha) = \min_x f_s(x)$. We can therefore select $\lambda_j^* = \alpha \geq 0, \forall j > n$. Further, the eigenvectors $\boldsymbol{u}_j \forall j \leq n$ lie in the range space of X and so $\boldsymbol{u}_j = X\boldsymbol{z}_j$ for some $\boldsymbol{z}_j$, for all $j \leq n$. Thus,

$$A^* = \sum_{j=1}^{n} \lambda_j^* \boldsymbol{u}_j^* \boldsymbol{u}_j^{*T} + \alpha \sum_{j=n+1}^{d} \boldsymbol{u}_j^* \boldsymbol{u}_j^{*T}$$
$$= \sum_{j=1}^{n} (\lambda_j^* - \alpha) \boldsymbol{u}_j^* \boldsymbol{u}_j^{*T} + \alpha \sum_{j=1}^{d} \boldsymbol{u}_j^* \boldsymbol{u}_j^{*T} = XSX^T + \alpha I.$$

Finally, we use the above two results to prove the main theorem. □

Proof. (Theorem 3.1.) Let $X = U_X \Sigma V_X^T$ be the SVD of X. Note that $K = X^T X = V_X \Sigma^2 V_X^T$, so $\Sigma V_X^T = V_X^T K^{1/2}$. Also, assuming $X \in \mathcal{R}^{d \times n}$ to be full-rank and $d > n$, $V_X V_X^T = I$.

Using Theorem A.3, the optimal solution to (2.1) is of the form $A = \alpha I + XSX^T = \alpha I + U_X \Sigma V_X^T S V_X \Sigma U_X^T = \alpha I + U_X V_X^T K^{1/2} S K^{1/2} V_X U_X^T = \alpha I + U_X V_X^T L V_X U_X^T$, where $L = K^{1/2} S K^{1/2}$. Thus, for spectral functions f, (2.1) is equivalent to (A.1). So using Theorem A.2, (2.1) is equivalent to (A.2) with $U = U_X V_X^T$ and $L = K^{1/2} S K^{1/2}$. Also, the losses can be simplified as

$$c_i(\alpha X^T X + X^T U L U^T X) \equiv c_i(\alpha K + K^{1/2} L K^{1/2}).$$

Let $K_A = \alpha K + K^{1/2} L K^{1/2} = \alpha K + KSK$. The theorem follows by substitution for L. □

The reader is directed to [35] for further details, as well as some corollaries to the above result.

Acknowledgments

I thank the anonymous reviewers for several suggestions, including pointers to recent results and ideas for improving the manuscript. I also thank Kilian Weinberger and Ruslan Salakhutdinov for figures that were reproduced in the manuscript. Further thanks go to Jason Davis for initially getting me interested in the metric learning problem, and to Michael Jordan for encouraging me to write this survey.

References

[1] A. Beck and M. Teboulle, "Mirror descent and nonlinear projected subgradient methods for convex optimization," *Operations Research Letters*, vol. 31, pp. 167–175, 2003.

[2] D. P. Bertsekas, *Nonlinear Programming*. Athena Scientific, 1999.

[3] M. Bilenko, S. Basu, and R. Mooney, "Integrating constraints and metric learning in semi-supervised clustering," in *Proceedings of International Conference on Machine Learning (ICML)*, 2004.

[4] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] L. Bottou, "Online algorithms and stochastic approximations," in *Online Learning and Neural Networks*, (D. Saad, ed.), Cambridge University Press, 1998.

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

[7] J. P. Boyle and R. L. Dykstra, "A method for finding projections onto the intersection of convex sets in Hilbert spaces," *Lecture Notes in Statistics*, vol. 37, pp. 28–47, 1986.

[8] L. M. Bregman, "The relxation method of finding the common points of convex sets and its application to the solutionof problems in convex programming," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 3, pp. 200–217, 1967.

[9] Q. Cao, Z. C. Guo, and Y. Ying, "Generalization bounds for metric and similarity learning," arXiv:1207.5437, 2012.

[10] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge University Press, 2006.

[11] R. Chatpatanasiri, T. Korsrilabutr, P. Tangchanachaianan, and B. Kijsirikul, "A new kernelization framework for Mahalanobis distance learning algorithms," *Neurocomputing*, vol. 73, no. 10–12, pp. 1570–1579, 2010.

[12] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "An online algorithm for lrage scale image similarity learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[13] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[14] P. Ciaccia, M. Patella, and P. Zezula, "M-Tree: An efficient access method for similarity search in metric spaces," in *Proceedings of International Conference on Very Large Data Bases (VLDB)*, 1997.

[15] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," in *Advances in Neural Information Processing Systems*, 2004.

[16] O. G. Cula and K. J. Dana, "3D texture recognition using bidirectional feature histograms," *International Journal of Computer Vision (IJCV)*, vol. 59, no. 1, pp. 33–60, 2004.

[17] Curious Labs, Inc., *Poser 5 — Reference Manual*. Santa Cruz, CA, 2002.

[18] H. Daume, "Frustratingly easy domain adaptation," in *Conference of the Association for Computational Linguistics (ACL)*, 2007.

[19] J. Davis and I. S. Dhillon, "Structured metric learning for high-dimensional problems," in *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[20] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, "Information-theoretic metric learning," in *Proceedings of International Conference on Machine Learning (ICML)*, 2007.

[21] R. Fletcher, "A new variational result for quasi-Newton formulae," *SIAM Journal on Optimization*, vol. 1, no. 1, 1991.

[22] J. Friedman, J. Bentley, and A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematics Software*, vol. 3, no. 3, pp. 209–226, 1977.

[23] A. Frome, Y. Singer, F. Sha, and J. Malik, "Learning globally consistent local distance functions for shape-based image retrieval and classification," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2007.

[24] T. Gaertner, "A survey of kernels for structured data," *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, 2003.

[25] A. Globerson and S. Roweis, "Metric learning by collapsing classes," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[26] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov, "Neighbourhood components analysis," in *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[27] A. A. Goldstein, "Convex programming in Hilbert space," *Bulletin of the American Mathematical Society*, vol. 70, pp. 709–710, 1964.

[28] G. H. Golub and C. F. Van Loan, *Matrix Computations*. Johns Hopkins University Press, 1996.

[29] K. Grauman and T. Darrell, "The pyramid match kernel: Efficient learning with sets of features," *Journal of Machine Learning Research*, vol. 8, pp. 725–760, 2007.

[30] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? Metric learning approaches for face identification," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2009.

[31] J. Ha, C. Rossbach, J. Davis, I. Roy, D. Chen, H. Ramadan, and E. Witchel, "Improved error reporting for software that uses black box components," in *Proceedings of Programming Language Design and Implementation (PLDI)*, 2007.

[32] S. C. H. Hoi, W. Liu, M. R. Lyu, and W. Y. Ma, "Learning distance metrics with contextual constraints for image retrieval," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006.

[33] G. Hua, M. Brown, and S. Winder, "Discriminant embedding for local image descriptors," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2007.

[34] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of Symposium on Theory of Computing (STOC)*, 1998.

[35] P. Jain, B. Kulis, J. Davis, and I. Dhillon, "Metric and kernel learning using a linear transformation," *Journal of Machine Learning Research*, vol. 13, pp. 519–547, 2012.

[36] P. Jain, B. Kulis, and I. Dhillon, "Inductive regularized learning of kernel functions," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.

[37] P. Jain, B. Kulis, I. Dhillon, and K. Grauman, "Online metric learning and fast similarity search," in *Advances in Neural Information Processing Systems (NIPS)*, 2008.

[38] P. Jain, B. Kulis, and K. Grauman, "Fast image search for learned metrics," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[39] W. James and C. Stein, "Estimation with quadratic loss," *Proceedings of Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 361–379, 1961.

[40] R. Jin, S. Wang, and Y. Zhou, "Regularized distance metric learning: Theory and algorithm," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[41] S. Kaski and J. Peltonen, "Informative discriminant analysis," in *Proceedings of International Conference on Machine Learning (ICML)*, 2003.

[42] D. Kedem, S. Tyree, K. Q. Weinberger, F. Sha, and G. Lanckriet, "Nonlinear metric learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[43] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2143–2157, 2009.

[44] B. Kulis, K. Saenko, and T. Darrell, "What you saw is not what you get: Domain adaptation using asymmetric kernel transforms," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

[45] B. Kulis, M. Sustik, and I. Dhillon, "Learning low-rank kernel matrices," in *Proceedings of International Conference on Machine Learning (ICML)*, 2006.
[46] B. Kulis, M. Sustik, and I. Dhillon, "Low-rank kernel learning with Bregman matrix divergences," *Journal of Machine Learning Research*, vol. 10, pp. 341–376, 2009.
[47] J. Kwok and I. Tsang, "Learning with idealized kernels," in *Proceedings of International Conference on Machine Learning (ICML)*, 2003.
[48] G. Lebanon, "Metric learning for text documents," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 497–508, 2006.
[49] E. S. Levitin and B. T. Polyak, "Constrained minimization problems," *USSR Computational Mathematics and Mathematical Physics*, vol. 6, pp. 1–50, 1966.
[50] H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristanini, and C. Watkins, "Text classification using string kernels," *Journal of Machine Learning Research*, vol. 2, pp. 419–444, 2002.
[51] P. C. Mahalanobis, "On the generalized distance in statistics," *Proceedings of the National Institute of Sciences of India*, vol. 2, no. 1, pp. 49–55, 1936.
[52] B. McFee and G. Lanckriet, "Metric learning to rank," in *Proceedings of International Conference on Machine Learning ICML*, 2010.
[53] G. J. McLachlan, *Discriminant Analysis and Statistical Pattern Recognition*. Wiley Interscience, 2004.
[54] S. Parameswaran and K. Q. Weinberger, "Large margin multi-task metric learning," in *Advances in Neural Information Processing Systems (NIPS)*, 2010.
[55] R. Rosales and G. Fung, "Learning sparse metric via linear programming," in *Proceedings of SIGKDD Conference*, 2006.
[56] S. Roweis, "EM algorithms for PCA and SPCA," in *Advances in Neural Information Processing Systems*, 1998.
[57] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," in *Proceedings of European Conference on Computer Vision (ECCV)*, 2010.
[58] R. Salakhutdinov and G. Hinton, "Learning a nonlinear embedding by preserving class neighbourhood structure," in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.
[59] B. Schoelkopf and A. Smola, *Learning with Kernels*. MIT Press, 2002.
[60] B. Schoelkopf, A. Smola, and K.-R. Mueller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998.
[61] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," in *Advances in Neural Information Processing Systems (NIPS)*, 2003.
[62] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter-sensitive hashing," in *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2003.
[63] S. Shalev-Shwartz, Y. Singer, and A. Y. Ng, "Online learning of pseudometrics," in *Proceedings of International Conference on Machine Learning (ICML)*, 2004.
[64] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[65] M. Slaney, K. Q. Weinberger, and W. White, "Learning a metric for music similarity," in *International Symposium on Music Information Retrieval (ISMIR)*, 2008.

[66] N. Snavely, S. Seitz, and R. Szeliski, "Photo tourism: Exploring photo collections in 3D," in *Proceedings of ACM SIGGRAPH*, 2006.

[67] I. Takeuchi, M. Nakagawa, and M. Seto, "Metric learning for DNA microarray data analysis," in *Proceedingso of International Workshop on Statistical-Mechanical Informatics (IW-SMI)*, 2009.

[68] M. Taylor, B. Kulis, and F. Sha, "Metric learning for reinforcement learning agents," in *Proceedings of International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2011.

[69] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society, Series B*, vol. 58, no. 1, pp. 267–288, 1996.

[70] M. E. Tipping and C. M. Bishop, "Probabilistic principal component analysis," *Journal of Royal Statistical Society, Series B*, vol. 21, no. 3, pp. 611–622, 1999.

[71] L. Torresani and K. Lee, "Large margin component analysis," in *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[72] D. Tran and A. Sorokin, "Human activity recognition with metric learning," in *Proceedings of European Conference on Computer Vision (ECCV)*, 2008.

[73] T. Tuytelaars and K. Mikolajczyk, "Local invariant feature detectors: A survey," *Foundations and Trends in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008.

[74] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters*, vol. 40, pp. 175–179, 1991.

[75] M. Varma and A. Zisserman, "A statistical approach to material classification using image patch exemplars," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 11, pp. 2032–2047, 2009.

[76] K. Q. Weinberger, J. Blitzer, and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," in *Advances in Neural Information Processing Systems (NIPS)*, 2005.

[77] K. Q. Weinberger and L. K. Saul, "Fast solvers and efficient implementations for distance metric learning," in *Proceedings of International Conference on Machine Learning (ICML)*, 2008.

[78] 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.

[79] K. Q. Weinberger and G. Tesauro, "Metric learning for kernel regression," in *Proceedings of International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2007.

[80] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," in *Advances in Neural Information Processing Systems (NIPS)*, 2002.

[81] H. Xiong and X. Chen, "Kernel-based distance metric learning for microarray data classification," *BMC Bioinformatics*, vol. 7, p. 299, 2006.

[82] Y. Ying, K. Huang, and C. Campbell, "Sparse metric learning via smooth optimization," in *Advances in Neural Information Processing Systems (NIPS)*, 2009.

[83] Y. Ying and P. Li, "Distance metric learning with eigenvalue optimization," *Journal of Machine Learning Research*, vol. 13, pp. 1–26, 2012.

[84] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proceedings of International Conference on Machine Learning (ICML)*, 2003.