Beyond Linear Similarity Function Learning

Julien Bohné

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science

University College London

October 7, 2016

I, Julien Bohné, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

Abstract

Being able to measure the similarity between two patterns is an underlying task in many machine learning and data mining applications. However, handcrafting an effective similarity function for a specific application is difficult and tedious. This observation has led to the emergence of the topic of similarity function learning in the machine learning community. It consists in designing algorithms that automatically learn a similarity function from a set of labeled data. In this thesis, we explore advanced similarity function concepts: local metric, deep metric learning and computing similarity with data uncertainty.

Linear metric learning is a widely used methodology to learn a similarity function from a set of similar/dissimilar example pairs. Using a single linear metric may be a too restrictive assumption when handling heterogeneous datasets. Lately, local metric learning methods have been introduced to overcome this limitation. However, most methods are subject to constraints preventing their usage in many applications. For example, some require the knowledge of all possible class labels during training. In this thesis, we present a novel local metric learning method, which overcomes some limitations of previous approaches.

Deep learning has become a major topic in machine learning. Over the last few years, it has been successfully applied to various machine learning tasks such as classification or regression. In this thesis, we illustrate how neural networks can be used to learn similarity functions which surpass linear and local metric learning methods.

Often, similarity functions have to deal with noisy feature vectors. In this context, standard similarity learning methods may result in unsatisfactory performance. In this thesis, we propose a method which leverages additional information on the noise magnitude to outperform standard methods.

Acknowledgements

I would like first to thank my supervisor, Massimiliano Pontil, for his precious advises and his patience. This PhD has been supported by a company named Safran Identity & Security and I want to thank Stéphane Gentric, the head of face recognition research, for his time and our fruitful discussions.

I am also immensely grateful to my wife, Véronique, for being the source of countless moments of happiness and supporting me in hard times.

Contents

List of Figures

List of Tables

Introduction

Pattern recognition tasks, such as clustering or nearest-neighbor classification, require computing similarity between pairs of data points. The overall task performance strongly depends on how well the similarity measure accurately reflects the actual proximity between the concepts represented by those points.

For many pattern recognition tasks the processing pipeline starts with a feature extraction step during which the raw data (e.g. images, speech signal, health monitoring) are transformed into feature vectors of fixed length. In this work, we focus on similarity functions for feature vectors in $\mathbb{R}^n$. In this setting, a similarity function is a function $f : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which outputs small values for pairs of similar inputs and larger values for dissimilar ones. In this thesis, in an abuse of terms we use *similarity function* to denote both similarity and dissimilarity functions (such as distances).

What *similar* means is entirely application dependent. For example, to obtain good performance in letter classification using a nearest-neighbor approach, the similarity between two images depicting the same letter should be high and it should be low for pairs of images of different letters. Alternatively, if the goal is to differentiate writing styles, the similarity measure should be completely different (See Figure 1). The Euclidean distance and other non-parametric similarity functions usually fail to deliver good performances because they ignore both the variable relevance of each feature for the task and the dependencies among those features.

Handcrafting a similarity function for a specific task may be complex and requires sophisticated prior knowledge of both the data and the task. However, it is often more effective and efficient to automatically learn the similarity function from a set of labeled data. Labels can be the class to which each data point belongs to, but several similarity function learning methods use a weaker form of supervision. They only need to be provided with a set of data pairs labeled *similar* when the two points belong to the

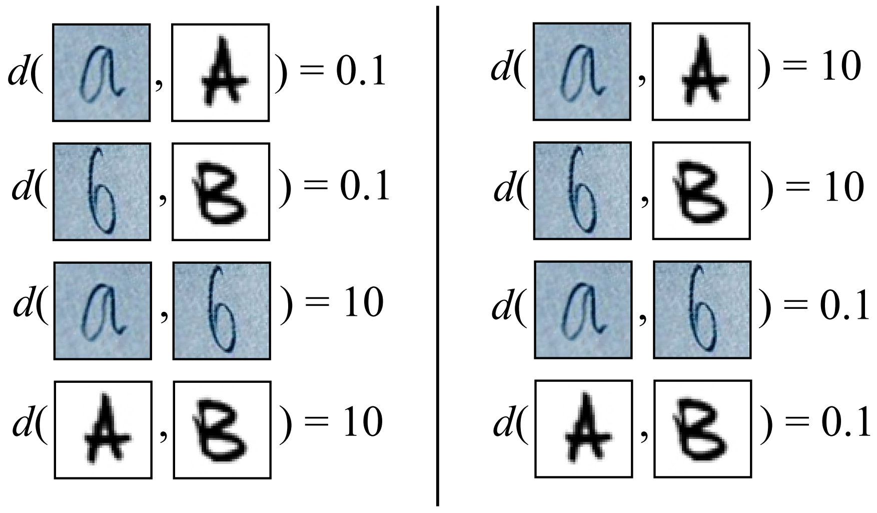


Figure 1: What a good similarity function is depends on the task of interest. On left side, $d(\cdot,\cdot)$ would be a desirable distance for letter recognition. On the right side, it would be a suitable distance for writing style comparison.

same class, or *dissimilar* when then do not. It is also popular to use triplets instead of pairs, each triplet containing two data points of the same class and a third point from a different class.

Learning a similarity function is a difficult task. Compared to other machine learning tasks, such as classification or regression, the need to consider the data points by pairs is a source of complexity as it belies the common assumption that training points are independent and identically distributed. Moreover, the number of parameters to learn grows quadratically with n. This tends to make similarity learning algorithms slower and relatively prone to over-fitting.

Similarity function learning has been a topic of interest in machine learning for many years but its popularity has increased recently when one of its most prominent applications, namely, face verification, became a central topic in computer vision. Face verification could be decomposed into three steps (see Figure 2). First, each face is detected and aligned (usually by finding a geometric transformation which puts points such as eyes, mouth corners etc. at specific locations). Second, the aligned image is transformed into a feature vector by applying filter banks, computing histograms of gradients etc. Then, a similarity score can be computed between pairs of feature vectors. To obtain the best overall performance, a good similarity function must be used and this is the topic of this thesis.

This PhD has been supported by Safran Identity & Security which is a company

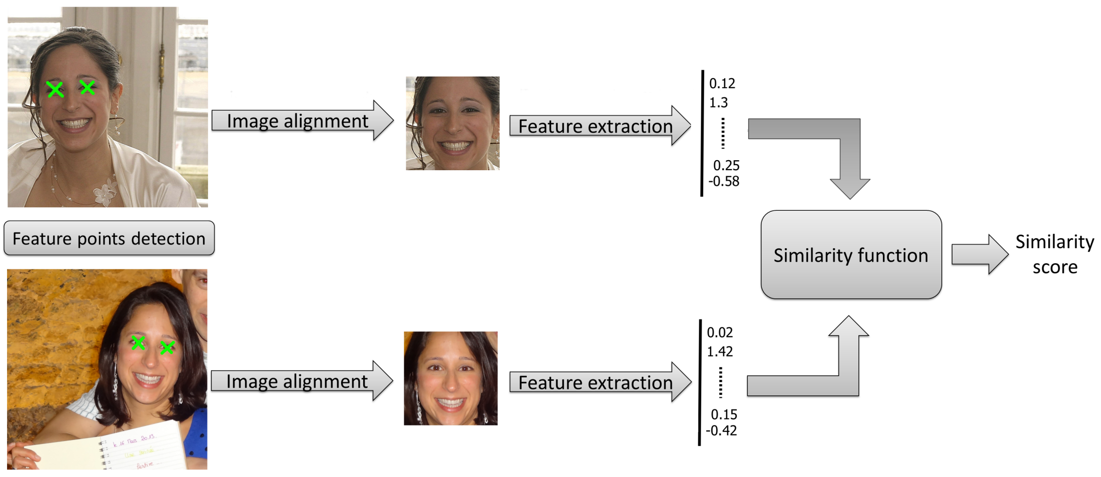


Figure 2: Face verification process

Figure 3: Border crossing using face recognition

designing biometric identification systems. For example, it provides automatic solutions based on face recognition to check at the border that a passport holder is its rightful owner (see Figure 3).

In Chapter 1, we give a general presentation about similarity functions and briefly describe the most popular methods. In Chapter 2, we describe the datasets used in this thesis to assess the performance of similarity functions. The training of most similarity function learning methods consists in optimizing an objective function. We discuss the components of such functions and related optimization methods in Chapter 3. Linear metrics obtain good results on a large variety of scenarios but are sometimes too simple to deal with heterogeneous data. Local metric learning is one of the type of methods which have been introduced to overcome this limitation. We present a novel local metric learning method in Chapter 4. Deep Learning is a strong trend in Machine Learning. It is widely used to perform classification but can also be used to compute similarities. Chapter 5 describes how to create a similarity function based on the Deep Learning principles. Uncertainty is a central issue in machine learning because the features we work with are often corrupted by noise. In Chapter 6, we introduce a new similarity function which takes advantage of information on the uncertainty of the features of each data point. Finally, we conclude this thesis by exposing the future challenges in similarity function learning.

Notation

We introduce here notations which are used throughout this thesis.

Notation	Description
x_i	Data point in $\mathbb{R}^n$
y_i	Class of x_i
r_{ij}	Similar/dissimilar pair indicator, $r_{ij} = 1$ if $y_i = y_j$ and -1 otherwise
$\mathbb{S}^n_+$	The set of positive definite matrices of size n
$\mathcal{S}$	Set of similar pairs, $(i,j) \in \mathcal{S} \implies r_{ij} = 1$
$\mathcal{D}$	Set of dissimilar pairs, $(i,j) \in \mathcal{D} \implies r_{ij} = -1$
$\mathcal{T}$	Set of training pairs, $\mathcal{T} = \mathcal{S} \cup \mathcal{D}$
$\|\cdot\|_F$	Frobenius norm
$\mathrm{Tr}(\cdot)$	Trace of a matrix
$[z]_+$	Positive part operator: $[z]_+ = \max(0,z)$
$\mathcal{N}(\cdot \mid \mu, S)$	Probability distribution function of the multivariate Normal distribution of mean μ and covariance matrix S

Chapter 1

Background

This chapter presents the different types of similarity functions and the most prominent methods to learn their parameters. Metric, or distance, is a popular class of similarity functions. We first give its definition and then describe the state-of-the-art algorithms to train a type of parametric distance called Mahalanobis distance. In the last section of this chapter, we extend our presentation to non-distance similarity functions.

1.1 Definition of a Metric

A metric over a set $\mathcal{A}$ is a function $d : \mathcal{A} \times \mathcal{A} \mapsto \mathbb{R}^+$ which satisfies 3 properties:

1. Symmetry: $\forall a, b \in \mathcal{A},\ d(a,b) = d(b,a)$,
2. Identity: $d(a,b) = 0 \iff a = b$,
3. Triangular inequality: $\forall a,b,c \in \mathcal{A},\ d(a,b) + d(b,c) \geq d(a,c)$.

The Minkowski distance of order p is defined in $\mathbb{R}^n$ by $\left(\sum_{k=1}^{n} |a_k - b_k|^p\right)^{1/p}$ where a_k and b_k are respectively the kth component of a and b. This family includes many of the usual non-parametric distances such as the Euclidean distance ($p = 2$), the Manhattan distance ($p = 1$) or the Chebyshev distance ($p = \infty$). Minkowski distances of order p with $p < 1$ are not metrics as they do not satisfy the triangular inequality, they might nonetheless be effective similarity functions for some applications.

Non-parametric distances cannot be adapted to take into account the specificities of the data and the task of interest. Therefore, they give poor results in practice when all features are not equally relevant for the task. Parametric similarity functions have been developed to overcome this limitation. In the remaining part of this chapter, we present the most popular types of similarity functions and the algorithms which have been proposed to learn their parameters.

1.2 Mahalanobis Distance

Originally, the Mahalanobis distance [1] has been introduced to deal with features of different scales and potentially correlated. The Mahalanobis distance between x_i and x_j in $\mathbb{R}^n$ is defined by $\sqrt{(x_i - x_j)^\top S^{-1}(x_i - x_j)}$ where S is an estimate of the covariance matrix of the data. By an abuse of language in the metric learning community, *Mahalanobis distance* is generally used to denote any distance of the form $d_M(x_i,x_j) = \sqrt{(x_i - x_j)^\top M(x_i - x_j)}$ where $M \in \mathbb{S}^n_+$, the cone of positive-semidefinite (PSD) matrices of size n. When M is not positive-definite, the Mahalanobis distance is not a distance but only a *pseudo-distance* because it does not satisfy the 2nd property but only weaker one which is $x_i = x_j \implies d(x_i,x_j) = 0$. Many metric learning algorithms have been proposed to find an appropriate matrix M given a set of labeled data, several are presented below.

Any PSD matrix M can be factorized as $M = L^\top L$ with $L \in \mathbb{R}^{n\times n}$. So, a Mahalanobis distance parametrized by M can be written as the Euclidean distance after applying the linear transformation L to the data: $d^2_M(x_i,x_j) = (x_i - x_j)^\top M(x_i - x_j) = (Lx_i - Lx_j)^\top(Lx_i - Lx_j)$. This equivalence is very useful to speed-up distance computation when each data point is compared to many others. Instead of computing $\sqrt{(x_i - x_j)^\top M(x_i - x_j)}$ for each pair, the linear transform is applied once per data point and Mahalanobis distances are replaced by Euclidean distance which is much faster. Some metric learning methods aim to find the best matrix M, whereas others try to find L directly. To further speed-up the distance computation and also reduce the memory needed to store the feature vectors, it can be interesting to reduce the feature vector dimensionality. Discriminative dimensionality reduction can be achieved by searching for a rectangular matrix $L \in \mathbb{R}^{r\times n}$ with $r < n$, such that the Euclidean distance in the reduced space is a good similarity function.

A popular choice for the matrix M comes from assuming that all classes follow multivariate Normal distributions and share the same covariance matrix [2]. This within-class covariance matrix is generally approximated by the within-class scatter matrix

$$S_w = \frac{1}{C}\sum_{c=1}^{C}\frac{1}{n_c}\sum_{i=1}^{n_c}\left(x_{c,i} - \frac{1}{n_c}\sum_{j=1}^{n_c} x_{c,j}\right)^\top\left(x_{c,i} - \frac{1}{n_c}\sum_{j=1}^{n_c} x_{c,j}\right) \tag{1.1}$$

where C is the number of classes in the training set and n_c the number of data points in the cth class. The probability that two data points x_i and x_j in $\mathbb{R}^n$ both belong to a class whose center is $\frac{x_i+x_j}{2}$ is equal to

$$\mathcal{N}\left(x_i \,\middle|\, \frac{x_i+x_j}{2}, S_w\right)\mathcal{N}\left(x_j \,\middle|\, \frac{x_i+x_j}{2}, S_w\right) = \frac{\exp\left(\frac{-(x_i-x_j)^\top S_w^{-1}(x_i-x_j)}{4}\right)}{2\pi\det(S_w)^n} \tag{1.2}$$

where $\mathcal{N}(\cdot \mid \mu, S)$ is the probability distribution function of the multivariate Normal distribution of mean μ and covariance S. Using the Mahalanobis distance with $M = S_w^{-1}$ is equivalent to using the above probability as a similarity function.

There is a tight link between using a Mahalanobis with $M = S_w^{-1}$ and the multi-dimensional Linear Discriminant Analysis (LDA) [3] which is a discriminative dimensionality reduction method. In several algorithms, such as FisherFace [4], the similarity function used is the Euclidean distance in the post-LDA subspace. The LDA finds a linear subspace which simultaneously maximizes the scattering of the class centers while making the classes as compact as possible. It leads to the optimization of the Fisher's criterion:

$$\max_{L\in\mathbb{R}^{m\times n}} \frac{\det\left(L^\top S_b L\right)}{\det\left(L^\top S_w L\right)} \tag{1.3}$$

where S_w is the within-class scatter matrix defined above and S_b is the scatter matrix of the class centers. The Fisher's criterion is invariant with regards to the norm of the columns of L whereas those norms have an impact on the performance of the Euclidean distance in the post-LDA subspace. The most common method to maximize the Fisher's criterion is to use the matrix whose columns are the generalized eigenvectors corresponding to the m largest eigenvalues in $S_b l = \lambda S_w l$. The resulting matrix L satisfies the equation $L^\top S_w L = I$ which is equivalent to $M = L^\top L = S_w^{-1}$ when $m = n$ (no dimensionality reduction). The between-class scatter matrix S_b has an impact only when the number of dimensions is effectively reduced: $m < n$.

The Mahalanobis distance with $M = S_w^{-1}$ only uses the within-class scatter matrix and discards all information about between-class variations. Köstinger et al. propose to use both with KISSME (Keep It Simple and Straightforward MEtric) [5]. This method takes as inputs a set of similar pairs and a set of dissimilar ones, respectively denoted $\mathcal{S}$ and $\mathcal{D}$, and assumes that feature vector differences of similar and dissimilar pairs follow 0-mean multivariate Normal distributions. The covariance matrices of those distributions, respectively denoted S_w and S_b, are simply estimated by the empirical covariance matrices of the feature vector differences of the corresponding pair set. When the data follows a statistical model, the likelihood ratio test is known to achieve the minimal error rate to choose between two mutually exclusive hypotheses (Neyman-Pearson lemma). We are interested in finding out if the difference between two feature vectors comes from a similar or a dissimilar pair, so we look at the likelihood ratio $\frac{P(x_i - x_j \,|\, \text{dis})}{P(x_i - x_j \,|\, \text{sim})}$. Given the Gaussian assumption presented above, the log-likelihood ratio is equal to $(x_i - x_j)^\top \left(S_s^{-1} - S_d^{-1}\right)(x_i - x_j)$ up to a constant. To obtain the closest pseudo-metric to this log-likelihood ratio, the authors take $M = \left(S_s^{-1} - S_d^{-1}\right)_+$ where $(\cdot)_+$ is the projection on the PSD cone operator.

Many metric learning algorithms are not based on a statistical modeling of the data but directly optimize an empirical objective function. The first method which has formulated metric learning this way is Mahalanobis Metric for Clustering (MMC) [6]. The matrix M is found by solving the following optimization problem:

$$\max_M \sum_{(i,j)\in\mathcal{D}} d_M(x_i - x_j) \tag{1.4}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{S}} d_M^2(x_i - x_j) \leq 1 \tag{1.5}$$

$$M \succ 0. \tag{1.6}$$

The optimization is performed by projected gradient ascent. After each gradient step, the matrix M is alternatively projected on the set of matrices satisfying each constraint until both are satisfied. The problem is convex and the procedure is therefore guaranteed to converge to the global optimum. This optimization method requires numerous eigenvalue decompositions of the matrix M and hence becomes rather slow when the dimensionality of the feature space increases.

Large Margin Nearest Neighbor (LMNN) [7] has been designed to improve nearest neighbor classification using a set of labeled points (x_i, y_i). The first step is to build a similar pair set $\mathcal{S}$ by selecting for each point the k-nearest neighbors of its class with the Euclidean distance. The second step is to optimize a cost function which gives a penalty when the squared distance of a similar pair (x_i, x_j) is larger than the squared distance between x_i and any point x_k belonging to a different class. To prevent infinite space inflation, the cost function also penalizes the sum of squared distances of similar pairs. More specifically, the cost function is given by

$$\min_{M\in\mathbb{S}_+^n} \sum_{(i,j)\in\mathcal{S}} d_M(x_i - x_j) + c \sum_{\substack{(i,j,k)\\ y_i=y_j\\ y_i\neq y_k}} \left[1 + d_M(x_i - x_j) - d_M(x_i - x_k)\right]_+ \tag{1.7}$$

where $[\cdot]_+ = max(0,\cdot)$ and c is a positive constant set by cross-validation. This optimization problem is convex and close to the standard SVM for classification problem. The authors propose an efficient optimization algorithm which takes advantage of the fact that most of the triplets (x_i, x_j, x_k) do not create any penalty. Several extensions of this work have been introduced to deal with multi-tasks learning [8], nonlinear projections [9] or local metrics [10]. More details about the latter are given in the next section.

In Logistic Discriminative Metric Learning (LDML) [11], the authors propose to cast the metric learning problem into a log-likelihood maximization. The squared Mahalanobis distance is transformed into the probability that the two points belong to the same class using a sigmoid function: $p_{ij} = P\left(y_i = y_j \,|\, x_i, x_j; M, b\right) = \left(1 + \exp\left(d_M(x_i, x_j) - b\right)\right)^{-1}$ where b is a bias term. The log-likelihood of the dataset is

$$\sum_{(i,j)\in\mathcal{S}} \ln p_{ij} + \sum_{(i,j)\in\mathcal{D}} \ln(1 - p_{ij}) \tag{1.8}$$

and is maximized with respect to M and b using a simple gradient ascent. This problem being concave, the optimization is guaranteed to converge to the global maximum.

Like with most machine learning tasks, over-fitting is a key issue in metric learning. Several articles propose a regularized objective function to limit over-fitting. In Information Theoretic Metric Learning (ITML) [12], the regularization is performed by penalizing the LogDet divergence between M and a given matrix M_0 which can be, for example, the identity matrix or the inverse of the sample covariance matrix. The cost function favors matrices M for which similar pair distances are below a given upper bound $u \in \mathbb{R}^+$ and dissimilar pair distances are above a given lower bound $\ell \in \mathbb{R}^+$:

$$\min_{M\in\mathbb{S}_+^n, \xi} \sum_{(i,j)\in\mathcal{S}} \left(\frac{\xi_{ij}}{u} + \log\frac{\xi_{ij}}{u}\right) + \sum_{(i,j)\in\mathcal{D}} \left(\frac{\xi_{ij}}{\ell} + \log\frac{\xi_{ij}}{\ell}\right) + \lambda d_{\ell d}(M, M_0) \tag{1.9}$$

$$\text{s.t. } d_M^2(x_i,x_j) \leq \xi_{ij} \qquad \forall (i,j) \in \mathcal{S} \tag{1.10}$$

$$d_M^2(x_i,x_j) \geq \xi_{ij} \qquad \forall (i,j) \in \mathcal{D} \tag{1.11}$$

where $d_{\ell d}(M,M_0) = \text{Tr}\left(MM_0^{-1}\right) - \log\det\left(MM_0^{-1}\right) - n$ is the LogDet divergence between two matrices and λ controls the strength of the regularization. This optimization problem is convex and the authors present an algorithm based on Bergman projections to solve it. There is no need to enforce the positive semi-definiteness of M specifically as this is automatically taken care of by the regularization term because $d_{\ell d}(M,M_0) = \infty$ if M and M_0 do not share the same range. A kernelized version and an online version of the algorithm are also proposed in [12].

Several results exist on generalization error bounds for metric learning. In [13], Jin et al. use uniform stability to show that, under some assumptions, a regularizer bounding $\text{Tr}(M)$ improves robustness with high dimensional data. A trace norm regularizer is also used in BoostMetric [14], a method which builds the matrix M by incrementally adding rank-1 matrices in a way similar to what Adaboost does with weak learners. Nonetheless, Maurer advocates in [15] that trace norm regularization can lead to too sparse solutions and learning instability. On the other hand, he shows a generalization error bound based on Rademacher complexity which depends on $\|M\|_F$. A metric learning algorithm is derived from this observation, it combines a hinge loss-based empirical error and a penalization of the Frobenius norm of matrix M, namely

$$\min_{M\in\mathbb{S}^n_+} \sum_{(i,j)\in\mathcal{S}\cup\mathcal{D}} \left[1 + \frac{r_{ij}}{\gamma}\left(1 - d_M^2(x_i,x_j)\right)\right]_+ + \lambda\|M\|_F \tag{1.12}$$

where γ is the width the margin and λ controls the regularization strength. More details about this objective function are given in Section 3.1.2. The optimization problem (1.12) is solved via stochastic gradient descent (SGD).

1.3 Other Types of Similarity Functions

As seen in the previous section, Mahalanobis distance learning has been widely addressed by the research community. However, other types of similarity function have also been explored. We present more complex function types in this section.

In some applications, it has been observed that data point norm contains no discriminative information. For example, when data points are obtained by applying a bank of linear filters on an image, the norm varies strongly with the image brightness and contrast. Therefore, the norm is an irrelevant information for many similarity function learning applications where invariance to such perturbations is sought. The cosine similarity $CS(x_i,x_j) = \frac{x_i^\top x_j}{\|x_i\|_2 \|x_j\|_2}$ is often used to obtain norm-invariant similarity functions. Like the Euclidean distance, the cosine similarity can also be parametrized by a PSD matrix M to adapt to the data and task at hand:

$$CS_M(x_i,x_j) = \frac{x_i^\top M x_j}{x_i^\top M x_i \, x_j^\top M x_j}. \tag{1.13}$$

Nguyen et al. apply this parametric cosine similarity to face verification [16]. They define the following objective function with respect to the linear data transformation matrix L

$$\max_L \sum_{(i,j)\in\mathcal{S}\cup\mathcal{D}} r_{ij} \, CS_M(x_i,x_j) + \lambda \, \|L - L_0\|_F^2 \tag{1.14}$$

where $M = L^\top L$, L_0 is a predefined matrix and λ controls the strength of the regularization. The regularization term is close in spirit to that of ITML. This optimization problem is not convex and there is no guarantee to reach the global maximum. The optimization procedure initializes the matrix L with L_0 and then uses the conjugate gradient method.

The Joint Bayesian method [17] relies on Gaussian assumptions over both the class centers distribution and that of the samples within a class. More specifically, each data point is expressed as $x_i = \mu_{y_i} + w_i$ where μ_{y_i} corresponds to the center of the class y_i and w_i to the variation with respect to this class center. Moreover, μ_{y_i} is assumed to be drawn from the distribution $\mathcal{N}(\cdot\,|\,0,S_\mu)$ and w_i from $\mathcal{N}(\cdot\,|\,0,S_w)$. The authors describe a variational Expectation-Maximization (EM) algorithm to estimate S_μ and S_w from a set of labeled training points (x_i,y_i). Given the model, it is possible to compute the likelihood of any data points pair (x_i,x_j) considering that they belong to the same class (H_{sim}) and the likelihood considering that they belong to different classes (H_{dis}). The authors propose to use the log-likelihood ratio

$$llr(x_i,x_j) = \log\left(\frac{P\left(x_i,x_j\,|\,H_{sim};S_\mu,S_w\right)}{P\left(x_i,x_j\,|\,H_{dis};S_\mu,S_w\right)}\right) \tag{1.15}$$

as similarity function. Thanks to the Gaussian assumptions, the log-likelihood ratio has a simple expression $llr(x_i,x_j) = x_i^\top M_1 x_j + x_i^\top M_2 x_i + x_j^\top M_2 x_j + const$ where M_1 and M_2 are two PSD matrices which are computed from S_μ and S_w. Other articles [18, 19] find the parameters of analogous similarity functions by optimizing hinge-loss based objective functions.

Using kernels is a common way to extend linear machine learning methods into non-linear ones [20] which has been popularized by SVMs. A non-linear similarity function can be created by combining a non-linear mapping $\Phi : \mathbb{R}^n \mapsto \mathcal{F}$ with a Mahalanobis distance in the feature space $\mathcal{F}$: $d^2_{M_\mathcal{F}}(x_i,x_j) = \left(\Phi(x_i) - \Phi(x_j)\right)^\top M_\mathcal{F} \left(\Phi(x_i) - \Phi(x_j)\right)$. However, the feature space $\mathcal{F}$ can be very high or even infinite-dimensional, making it costly or impossible to explicitly compute the mapping Φ. The kernel trick allows to implicitly make calculations in the feature space without actually performing the mapping. To make this possible we need to define a kernel $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to an inner product in the feature space $\mathcal{F}$. Conveniently, the Mercer's theorem states that any functions fulfilling the Mercer's condition is an inner product in some feature space. Nonetheless, it is not trivial to kernelize an algorithm as it requires expressing all operations on data points as inner products. The LDA presented in the previous section is one of the first methods which have be kernelized [21]. A non-linear version of ITML using kernels has also been proposed by its authors [12].

As we have already seen, there is a strong connection between learning a similarity function and learning a data transformation such that a non-parametric similarity like the Euclidean distance delivers good performance. The idea of using neural networks to perform the data transformation has first been explored in the 90's for signature verification [22] and more recently for face recognition with Convolutional Neural Network (CNN) [23, 24] or standard descriptor-based approaches [25, 26]. In all these methods, the weights of the neural network are obtained by minimizing an objective function similar to those used in ITML, LMNN, LDML, etc. The main differences are the inputs types (images or feature vectors) and the network structure.

Mahalanobis distance has been shown to be quite effective on various tasks but it suffers from a strong limitation: a single linear metric is used to compare data over all the input space. This may be inappropriate in order to handle heterogeneous data. This observation is the root of the development of local metric learning methods which adapt the dissimilarity function to the local specificities of the data. For illustrative purpose let us consider two examples. It is well known that in digit classification some digits are easily mistaken for another such as “1” and “7” or “3” and “8”. It seems therefore reasonable to focus on different features to discriminate digits in the “1-7” region and in the “3-8” one in order to reduce the number of misclassifications. Our second example is face verification: should we put the emphasis on the very same features to compare two pictures of Caucasian males and two pictures of Asian females? Weinberger and Saul proposed an extension of LMNN to local metric (MM-LMNN [10]), in which a specific metric is associated to each class and all metrics are jointly learned to optimize a classification criterion. KISSME [5] has also been extended to local metric in [27] where one KISSME metric is learned separately for each class. These class-specific metrics are averaged with a global one to limit the risk of over-fitting due to the fact that each metric might be learned using only a limited number of training samples. GLML [28] uses a local metric to optimize nearest neighbor classification using the class conditional probability distributions.

All these local metric learning methods suffer from the same drawback, namely they need enough training samples per class to estimate the metrics. Therefore, they cannot be employed directly in applications in which there is a large number of classes with only a few training data points in some classes. To overcome this problem, a local metric learning method is introduced in [29] based on finite a number of linear metrics named PLML. The number of metrics is different from the number of classes and hence the method can scale up to a larger number of classes. However, this method is specifically designed for nearest neighbors classification as it can only compute the similarity of pairs for which at least one data point is in the training set. This strongly limits the practical range of tasks that PLML can deal with. For example, it prevents the application of PLML to the problem of face verification. In Chapter 4, we introduce a method called LMLML which overcomes those limitations.

When several local metrics are learned, one issue is to determine how to compare data samples which come from different subparts of the input space. CLML [30] is an alternative to LMLML. It jointly learns many locally linear projections such that any pair of projected points can be effectively compared using Euclidean distance. Like in LMLML, the input space is soft-partitioned using a GMM. More details about this partitioning are given in Section 4.1.

Chapter 2

Performance Evaluation

In the next chapters, we compare the performance of several similarity function learning methods. We present here how we measure performances and the datasets on which those methods have been evaluated.

2.1 Performance Measures

In this section we present the performance measures we use to evaluate similarity functions in this thesis. A similarity function takes two inputs and outputs a score. Therefore, it is natural to evaluate it like a pair classifier with a DET curve that we define below. The process to obtain this curve is composed of four steps:

1. We compute similarity scores for a large number of pairs for which we know the correct labels (similar or dissimilar).
2. For every possible threshold t, we compute the number false positive errors (dissimilar pairs dimmed similar by thresholding the similarity) and false negative errors (similar pairs dimmed dissimilar by thresholding the similarity).
3. Those error counts are transformed into rates by dividing them by the corresponding number of pairs to obtain a False Positive Rate (FPR) and a False Negative Rate (FNR) for every threshold t.
4. We plot the FNR as a function of the FPR. This curve is called a Detection Error Tradeoff (DET) curve.

A DET curve displays the performance of a similarity function at all the possible operating points (thresholds) but for some applications, like face verification, people are often focused on performance at low FPR. To better visualize the performance at low FPR, a common practice is to plot the DET curve with the FPR axis in logarithmic scale.

A DET curve gives of global view of the performance of a similarity function but it is sometimes convenient to synthesize the performance in one value to compare different methods easily. The Equal Error Rate (EER) is often used to this end. It is defined by $\text{EER} = \text{FNR}(t)$ where the threshold t is chosen so that $\text{FPR}(t) = \text{FNR}(t)$. It is also common to compare methods by looking at the FNR at a given value of FPR. The chosen FPR depends on the operating point of interest for the target application.

Similarity functions are also used for classification tasks to improve performance of a k-nearest neighbor classifier. In this case, we usually measure the performance by looking at the classification accuracy which corresponds to the proportion of requests correctly classified.

2.2 Classification Datasets

Similarity function learning is often used as a means to improve nearest neighbors classification. Indeed, some of the most famous metric learning methods such as LMNN [7] have been specifically designed for this problem. The five datasets we used for classification are described below.

2.2.1 MNIST

Handwritten digits classification has been widely used to assess the performance of similarity functions for classification. The MNIST dataset is composed of 70000 images of size 28×28, 60000 for training and 10000 for testing. Figure 2.1 shows some examples of images composing the dataset. To create feature vectors, we do a PCA directly on the pixel values and 164 dimensions are kept after dimensionality reduction to retain 95% of the energy.

2.2.2 Isolet

Isolet is a spoken-letter datasets which contains 6238 vectors for training and 1559 for testing. 150 subjects were asked to pronounce each letter of the alphabet twice. The 617-dimensional feature vectors are composed of spectral coefficients, contour features, sonorant features, pre-sonorant features and post-sonorant features.

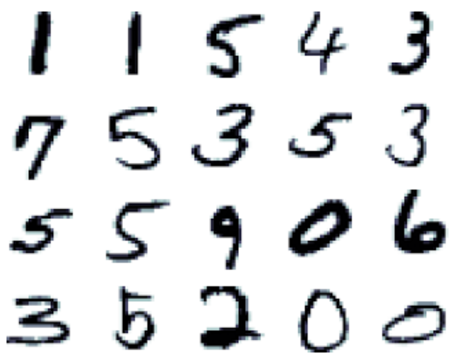


Figure 2.1: Samples from MNIST digits

2.2.3 Letter

Letter is a typed-letter recognition dataset. Distortions have been applied to the letters from 20 different fonts to create 20000 16-dimensional feature vectors composed of various statistical cues (mean, variance, correlation etc.) and edge count. The 16000 first samples are used for training and the remaining 4000 for testing.

2.2.4 Reuters

The text categorization dataset Reuters-21578 R52 consists of 9100 text documents which appeared on the Reuters newswire in 1987, 6532 in the training test and 2568 in the testing set. Each text belongs to one of the 52 topics and every topic has at least one text in the training set and one in the testing set. The classes are very unbalanced, some topics have more than 1000 text documents whereas others have just a few. Each text is described by a histogram of word occurrence spanning 5180 terms. A very large number of dimensions should be kept after the PCA to preserve 95% of the energy but to speed up the experiments we kept only the first 100 dimensions retaining 62% of the energy.

2.2.5 20newsgroup

20newsgroup is composed of 18774 messages taken from 20 Usenet newsgroups spanning topics as diverse as alt.atheism or rec.motorcycles. Each text is represented by a 20000-dimensional word count histogram. The histograms are available for download on the Internet but each paper using this dataset proposes its own processing pipeline to transform those high dimensional histograms into more easily usable feature vectors (filtering with a stop-list, normalization, dimensionality reduction, etc.). We have implemented our own pipeline to create 1000-dimensional feature vectors.

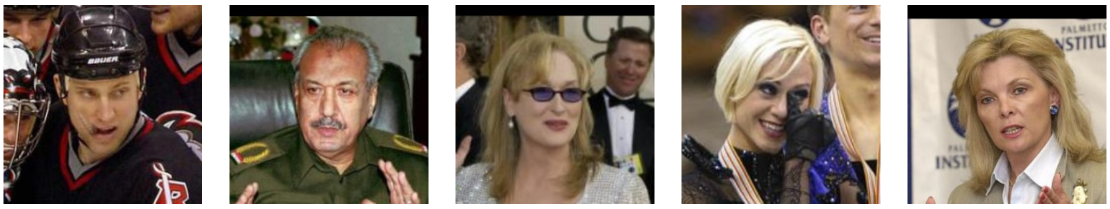


Figure 2.2: Images from LFW

2.3 Face Verification Datasets

Face verification is an important and practical application of similarity function learning. We present results on two face datasets: LFW and FRGC.

2.3.1 LFW

Labeled Faces in the Wild (LFW) [31] is a popular face verification dataset. It is composed of 13233 images of 5749 people taken from *Yahoo! News* in a wide range of acquisition conditions (pose, illumination, expression, age, etc.). Some examples are given in Figure 2.2. The best currently published results on LFW [24, 32] are obtained with deep learning methods trained on very large outside datasets (up to several millions). In this thesis, we use an experimental setup where learning is performed on a small number of predefined training pairs (2700 similar and 2700 dissimilar) called *restricted setting*. We use a simple feature extractor which has been often used to assess performance of metric learning algorithms on LFW [16]. We start from the "aligned" images that we cropped to 150×80 to remove most of the background, then we extracted descriptors composed of histograms of Local Binary Patterns [33] and finally we reduced their dimensionality to 300 by PCA.

2.3.2 FRGC

FRGC Experiment 1 [34] is a face dataset of 15000 images from more than 500 people. Figure 2.3 shows some examples. The pose of the subjects and the illumination have been controlled during the acquisition. Compared to datasets like Labeled Faces in the Wild presented above, this dataset is fairly simple. However, on this type of dataset, the interest is focused on the verification rate at low false positive rates (1% and below). This is a realistic setting for many security applications of face recognition (like smartphone unlocking or passport check at the border) where a false acceptance is a security breach and therefore should be very rare. After aligning the images using

Figure 2.3: Images from FRGC

the eyes locations, we computed UoCTTI HOG descriptors [35] extracted using the *VLFeat* library [36] to obtain 6076-dimensional feature vectors. We then use PCA to reduce the dimensionality to 700.

Chapter 3

Objective Functions for Empirical Loss Minimization

Similarity function learning methods can be classified into two categories. First, methods based on a statistical model of the data (for example KISSME [5] or the Joint Bayesian Method [17]). Second, methods based on the optimization of an objective function which depends on a set of labeled training pairs (or triplets). This category includes many of the most popular metric learning methods such as ITML [12] or LMNN [7]. This type of approach is often considered more flexible and more effective as it makes no assumption on the data distribution. In this chapter, we first describe the two components of objective functions: the empirical loss and the regularization term. The former drives the optimization toward a solution which performs well on the training set while the latter aims at limiting over-fitting to obtain good generalization performance. We conclude this chapter by presenting an optimization method based on Stochastic Gradient Descent.

3.1 Empirical Loss

3.1.1 Linear Loss

When the dataset is composed of similar and dissimilar pairs, the goal of the learning is to get small distances for similar pairs and large distances for dissimilar ones. A simple objective function has been proposed in [37], yielding to the optimization problem

$$\min_{M\in\mathbb{S}^n_+} \sum_{(i,j)\in\mathcal{T}} r_{ij} \left(x_i - x_j\right)^\top M \left(x_i - x_j\right)$$

$$\text{s.t. } \|M\|_F \leq 1 \tag{3.1}$$

where r_{ij} is equal to 1 if (i, j) is a similar pair and -1 otherwise. The constraint on $\|M\|_F$ prevents that an increase of the scale of M systematically leads to a lower cost whereas the loss should be intrinsically invariant to the scale of M. The problem can be rewritten as

$$\begin{aligned} &\max_{M \in \mathbb{S}_+^n} \langle M, A \rangle \\ &\text{s.t. } \|M\|_F^2 \leq 1 \end{aligned} \tag{3.2}$$

where $A = -\sum_{(i,j)\in\mathcal{T}} r_{ij} \left(x_i - x_j\right) \left(x_i - x_j\right)^\top$ and $\langle \cdot, \cdot \rangle$ is the matrix inner product. We now prove that the solution to this problem is

$$M = \frac{A_+}{\|A_+\|_F} \tag{3.3}$$

where A_+ is the positive part of the matrix A.

Let $M = U\text{diag}(\lambda)U^\top$ and $A = V\text{diag}(\mu)V^\top$ be the eigendecompositions of matrices M and A respectively. The two constraints on M both depend only on its spectrum λ: $\|M\|_F^2 \leq 1 \iff \sum_k \lambda_k^2 \leq 1$ and $M \in \mathbb{S}_+^n \iff \forall k \in \{1, \dots, n\}, \lambda_k \geq 0$. Moreover, $\forall M \in \mathbb{R}^{n\times n}$ and $A \in \mathbb{R}^{n\times n}$, the inequality $\langle M, A \rangle \leq \langle \lambda, \mu \rangle$ holds and is tight if and only if $U = V$. Since we want to maximize $\langle M, A \rangle$ and that the constraint $\|M\|_F^2 \leq 1$ does not impact U, we can deduce that $U = V$. We now need to solve

$$\begin{aligned} &\max_{\lambda} \sum_{k=1}^{n} \lambda_k \mu_k \\ &\text{s.t. } \sum_{k=1}^{n} \lambda_k \leq 1 \\ &\quad \forall k \in \{1, \dots, n\}, \lambda_k \geq 0. \end{aligned} \tag{3.4}$$

which has a trivial solution:

$$\lambda_k = \frac{[\mu_k]_+}{\sqrt{\sum_{k'=1}^{n} [\mu_k]_+}^2} \tag{3.5}$$

The solution to the problem (3.2) is therefore $M = V\mathrm{diag}(\lambda)V^\top$ which is equivalent to $M = \frac{A_+}{\|A_+\|_F}$.

The weakness of the linear loss is that every pair equally contributes to the loss whereas some might carry more discriminative information than others. This explains the bad performance presented in the next section when compared, for example, to the hinge loss.

3.1.2 Hinge Loss

The hinge loss has been popularized by Support Vector Machine. As opposed to the linear loss, only the hard-to-classify training examples have an impact on the empirical loss. In the context of metric learning, we define our hinge loss by

$$\ell_\gamma\left(r_{ij}, z\right) = \left[1 - \frac{r_{ij}}{\gamma}(1-z)\right]_+ \tag{3.6}$$

where γ corresponds to the margin width and z is a dissimilarity between x_i and x_j. The parameter γ is typically between 0.5 and 1: only the most difficult pairs impact the objective function when γ is small but a larger proportion of them does if γ is large. Its optimal value depends on how helpful easy pairs are to improve the performance on the part of the DET curve we care about (low false positive or equal error rate for example). It also depends on the size of the training set: larger values of γ are better with small training sets because when few pairs are available it is better not to discard too many of them even if they are not the most helpful ones. ℓ_γ is plotted in Figure 3.1 for the two types of pairs. Equivalently, the hinge loss can be formulated with the help of slack variables and constraints:

$$\begin{aligned} \ell_\gamma\left(r_{ij}, z\right) &= \min \xi \\ \text{s.t. } z - \gamma + 1 &< \xi \qquad \text{if } r_{ij} = 1 \\ \gamma - z + 1 &< \xi \qquad \text{if } r_{ij} = -1 \\ \xi &\geq 0. \end{aligned} \tag{3.7}$$

With the hinge loss, learning a metric can be formulated as the following opti-

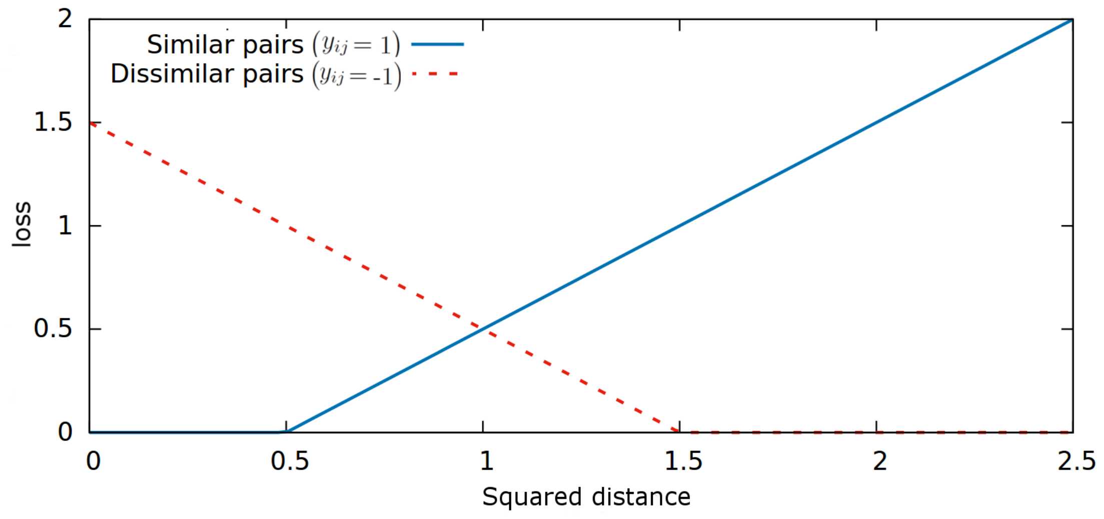


Figure 3.1: $\ell_{\gamma,b}(r,z)$ with $\gamma = 0.5$

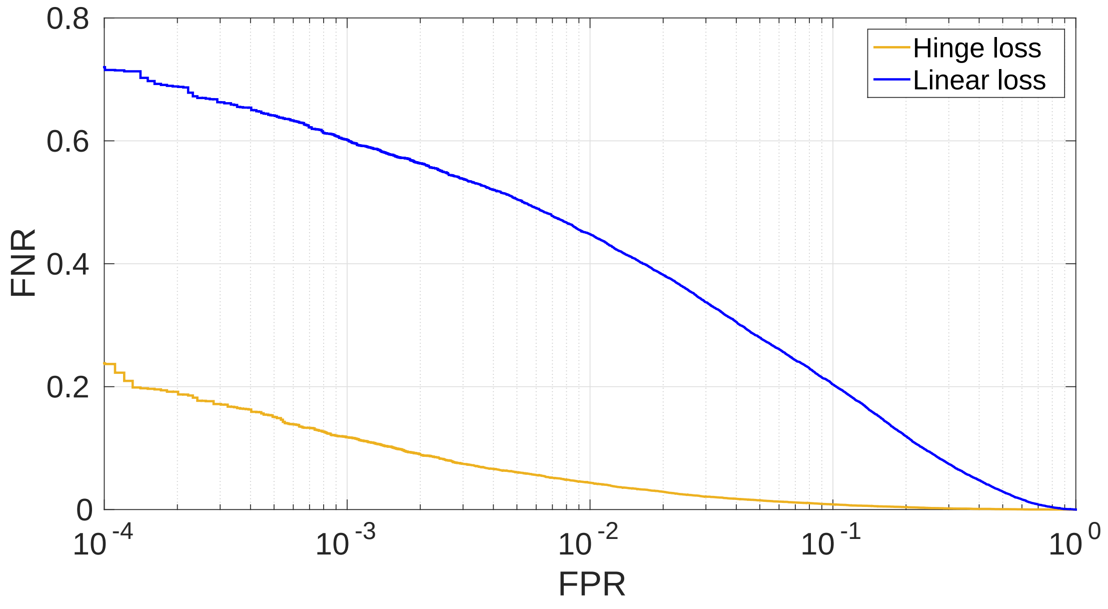


Figure 3.2: Performance comparison between the hinge loss and the linear loss on FRGC

mization problem:

$$\min_{M \in \mathbb{S}_+^n} \sum_{(i,j)\in\mathcal{T}} \ell_\gamma\left(r_{ij}, d_M^2(x_i,x_j)\right). \tag{3.8}$$

The use of the hinge loss leads to better similarity functions than the linear loss presented in the previous subsection (see Figure 3.2). The hinge loss focuses on difficult pairs whereas the linear loss also tries to move pairs which are already well classified and therefore have no real impact on the performance.

For the classification task, many loss functions have been proposed in the literature. The most frequently encountered are the exponential loss $e^{\frac{r_{ij}}{\gamma}(1-d_M^2(x_i,x_j))}$ used in AdaBoost [38], and the logistic loss $\log\left(1+e^{\frac{r_{ij}}{\gamma}(1-d_M^2(x_i,x_j))}\right)$ which comes from Logistic Regression. They can both be adapted to learn similarity functions and, being convex losses, do not create specific optimization difficulties. Those different loss functions usually obtain similar accuracy but depending on the dataset some might be more effective than others. In this thesis, we only used the hinge loss because it has the advantage of being faster to compute.

3.1.3 Data Preprocessing

As it is common in similarity function learning, we apply a data preprocessing step. This step serves two purposes: first it reduces the dimensionality to speed up computation for both training and testing, and second it reduces the noise thereby improving the overall performance of the algorithm.

As with most metric learning methods, we first center the dataset and reduce the dimensionality to a n-dimensional space by PCA; n is often chosen so that 95% of the energy is preserved. We compute the within-class scatter matrix defined by

$$S = U\left(\sum_{(i,j)\in\mathcal{S}}(x_i-x_j)(x_i-x_j)^\top\right)U^\top \tag{3.9}$$

where U is the matrix formed by the n leading eigenvectors of the covariance matrix of the data, and then multiply the data by $S^{-1/2}$ to make the classes isotropic on average. This transformation is known under different names such as *mapping in the intra-personal subspace* in face recognition [18] or *Within-Class Covariance Normalization (WCCN)* in the speaker recognition community [2]. The transformed data points are now $x' = S^{-1/2}U(x-m)$ where m is the mean of the data. Like in [18], we finally rescale each feature vector so that it has a unit $\ell2$-norm.

3.1.4 Learning from Pairs or Triplets?

A similarity function takes a pair of input points and outputs a similarity value. It is therefore quite natural to learn the function parameters using pairs of inputs as it reflects how the function will actually be used. Usually, loss functions based on pairs generate a cost if a similar pair has a distance larger than a given threshold θ_s or a dissimilar one a distance smaller than a second threshold θ_d. It therefore leads to a similarity function which can effectively be thresholded to take decisions. This property is very desirable for many applications, such as face verification for example. Some other similarity function learning methods use triplets of data points (x_i, x_j, x_k) with $y_i = y_j$ and $y_i \neq y_k$ and their objective function penalize small or negative values of $d^2(x_i, x_j) - d^2(x_i, x_k)$. These objective functions are not primarily designed to create similarity functions whose output can be thresholded but rather functions designed to answer questions such as "Is x_i more similar to x_j than to x_k?". Triplets-based methods are therefore better suited for ranking applications (e.g. nearest neighbor classification) than pair verification. In this thesis, we focus on pairwise objective functions because we are interested in both ranking and thresholding.

3.1.5 Selecting Training Pairs from Class Labels

In many applications, training pairs are not provided but have to be built from labeled data points. The number of potential pairs grows quadratically with the number of training points, therefore it is often impracticable to use all of them because the training would require a prohibitive amount of time. The choice of training pairs has a significant impact on the performance of the similarity functions learned.

When the similarity function is employed for nearest neighbors classification, it makes sense to pick training pairs composed of neighboring points because nearest neighbor classifiers base their decision only on such points. In this case, we propose proceeding as follow: for each data point x, create k similar pairs, formed by x and each of its k closest neighbors from the same class, and k dissimilar pairs, formed by x and each of its k closest neighbors from a different class. This results in $2k$ training pairs per data point. We use $k = 5$ in all our nearest neighbor classification experiments.

Similarity measures may also be thresholded to take a decision, as in face verification for example. The choice of the threshold leads to a specific trade-off between false positive and false negative rates. On datasets such as FRGC with few images per identity, the number of potential similar pairs is limited and all of them can be used during training but a selection has to be made for the dissimilar ones. To learn a similarity function performing well at low false positive rates, the training set should include a large number of dissimilar pairs. To speed up the training, we use a simple hard dissimilar pairs mining scheme. We randomly pick a number of dissimilar pairs equal to the number of similar pairs first and train our model with this set. We then compute the

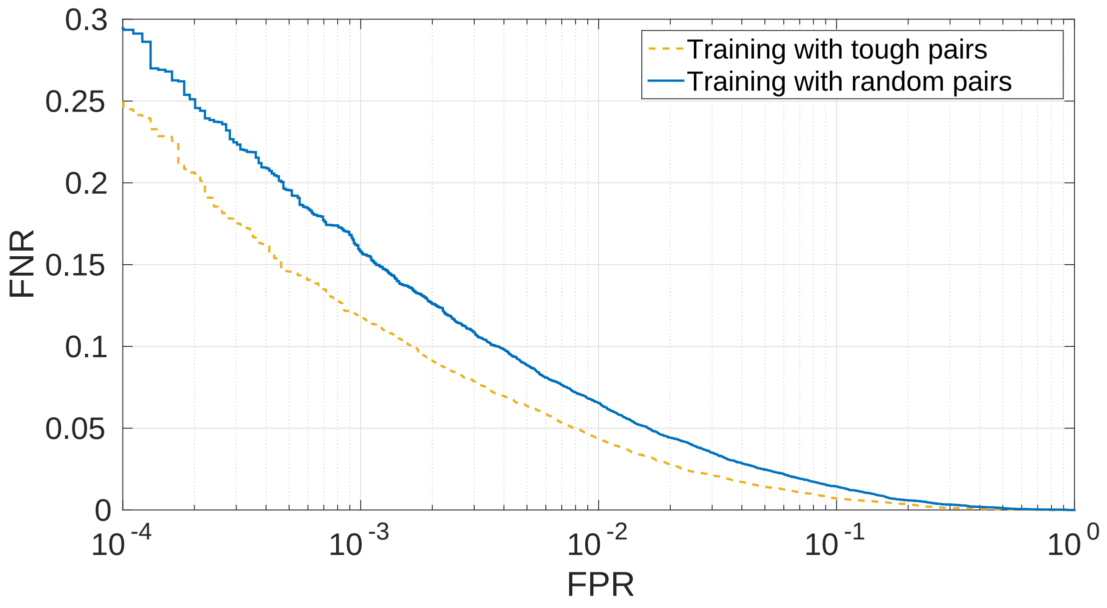


Figure 3.3: On FRGC, using tough dissimilar pairs improve performance at low false positive rates.

similarity for a large number of dissimilar pairs and select the 5 or 10% hardest pairs to train the similarity function again. This step could be repeated several times but in practice we observed little improvement after the first iteration. Figure 3.3 shows the improvement between learning the metric described in Section 3.1.2 with random and tough dissimilar pairs.

3.2 Common Metric Learning Regularizers

Over-fitting is often considered to be one of the main issues in machine learning. It occurs when the model to train is too complex, too powerful for the training data at our disposal. The consequence is that the learned model has a very low training error (error on the training data) but a high generalization error (error on unseen data). To limit over-fitting, one solution is to restrict the model complexity by limiting its number of parameters, for example using a low degree polynomial for curve fitting. The second possibility which offers more flexibility is to add a regularization term to the objective function. The regularizer penalizes complex models but they can nonetheless be chosen if the performance gain on the training set is large enough.

In the context of metric learning, the optimization problem combining the hinge loss (3.6) and a regularizer can be formulated as

$$\min_{M \in \mathbb{S}_+^n} \sum_{(i,j)\in\mathcal{T}} \ell_\gamma\left(r_{ij}, d_M^2(x_i,x_j)\right) + \lambda R(M) \tag{3.10}$$

where $R(M)$ is the regularizer and λ is a meta-parameter controlling the regularization strength which should be set by cross-validation. Several regularizers for metric learning have been proposed in the literature. To provide an intuition about regularizer for metric learning, we rewrite the squared Mahalanobis distance using the eigendecomposition $M = UDU^\top$:

$$d_M^2(x_i,x_j) = (x_i - x_j)^\top UDU^\top (x_i - x_j) \tag{3.11}$$

$$= \sum_{k=1}^{n} D_{k,k} \left(x_i^\top U_{\cdot,k} - x_j^\top U_{\cdot,k}\right)^2 \tag{3.12}$$

where $D_{k,k}$ is the kth diagonal element of D and $U_{\cdot,k}$ is the kth column of U. We see that $d_M^2(x_i,x_j)$ is a weighted sum of squared differences along orthogonal directions, the weights being the eigenvalues of the matrix M. If M has a very unbalanced spectrum, the overall distance depends mostly on a few terms of the sum. On the contrary, if the spectrum of M is flat, all terms contribute to $d_M^2(x_i,x_j)$ with comparable intensity. Domain specific knowledge can help to choose the suitable regularizer. If the discriminative information lies in a low dimensional subspace, then a regularizer favoring sparse spectrum should be chosen. Penalizing the rank of M leads to non-convex optimization problems. The common approach is to penalize the trace norm regularizer $\mathrm{Tr}(M)$ [13, 14] as it is the best convex approximation of the rank of M. On the other hand, if there is a prior that no direction in the feature space should out-weight the others, a regularizer favoring a flat spectrum should be preferred. The regularizer is meant to balance the fact that the empirical cost only takes into account the directions which matter in the training set. The Frobenius norm of the matrix M is a simple convex regularizer well suited in this case as $||M||_F = \sqrt{\sum_{k=1}^{n} D_{k,k}{}^2}$. This assertion is supported by the fact that a generalization error bound function of $||M||_F$ can be computed [15].

Figure 3.4, 3.5 and Table 3.1 show the performance on MNIST, Reuters, FRGC and LFW (see Chapter 2 for details about these datasets). We compare the Euclidean distance ($M = I$) with similarity functions learned without regularization and with the

	Classification Accuracy	
Method	MNIST	Reuters
Euclidean distance	96.61%	86.77%
Hinge loss without regularization	96.94%	88.02%
Hinge loss and trace norm regularizer	96.91%	88.04%
Hinge loss and Frobenius norm regularizer	**97.40%**	**88.64%**

Table 3.1: Classification Accuracy of hinge loss-based loss functions on MNIST and Reuters

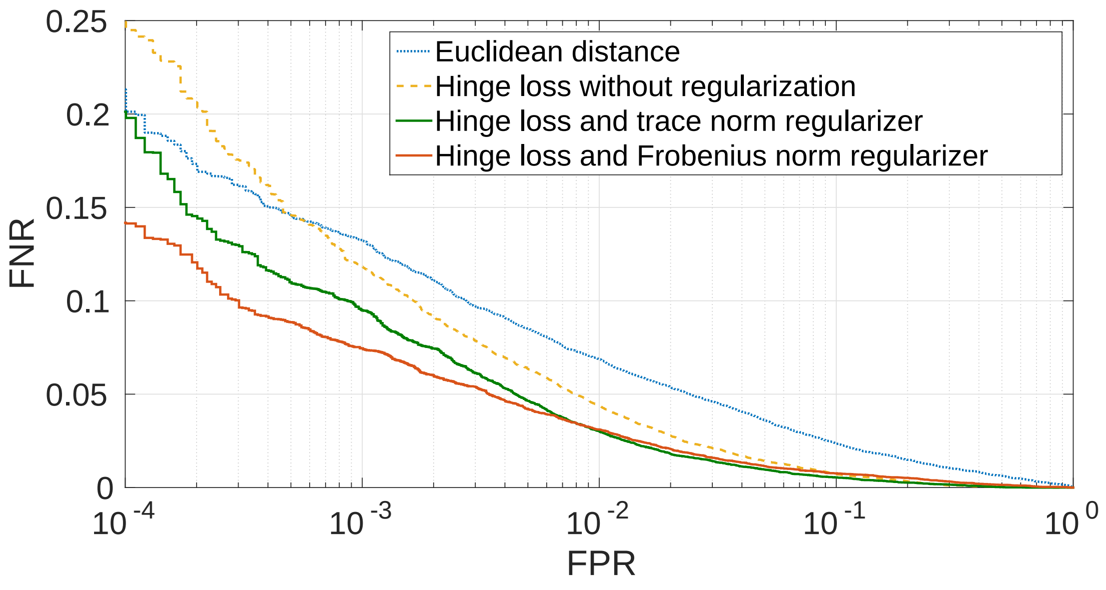


Figure 3.4: Performance of hinge loss-based loss functions on FRGC.

regularizers $\mathrm{Tr}(M)$ and $||M||_F$. For all the datasets, we feed similarity functions with data preprocessed as described in Section 3.1.3. Therefore, the Euclidean distance in this new space is equivalent to the cosine similarity with $M = S_w^{-1}$ in the original feature space. In our experiments, the discriminative information does not seem to be concentrated in a low-dimensional subspace and, therefore, the trace norm which induces too much sparsity in the spectrum of M leads to a worse performance than the Frobenius norm. On LFW, MNIST and Reuters, the trace norm regularizer has even a negative impact on the performance and therefore very small values for λ are selected by the cross-validation process. This explains why performance without regularization and with the trace norm regularizer are very close in Figure 3.5 and Table 3.1.

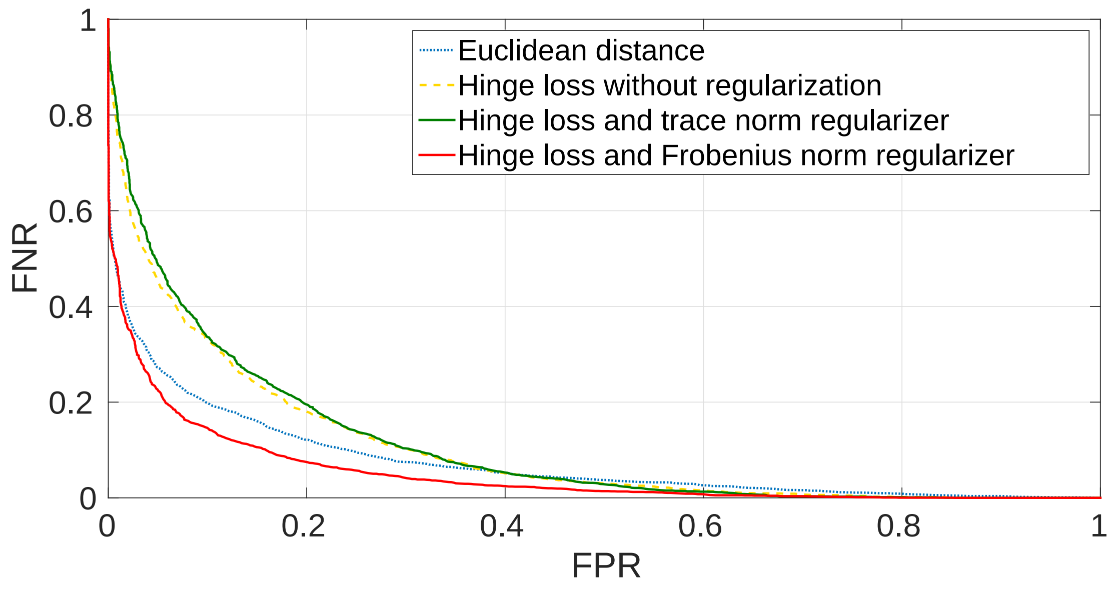


Figure 3.5: Performance of hinge loss-based loss functions on LFW.

3.3 A New Regularizer for Metric Learning

The Frobenius norm $||M||_F$ is the most common regularizer in metric learning, it often leads to good results but is not always optimal. In particular, if prior information that Euclidean distance is already a good metric is available, the Frobenius norm might not be the best choice. This occurs, for example, when feature vectors are preprocessed with a method such as that described in Section 3.1.3 because it aims at getting good results with the Euclidean distance which is equivalent to using $M = I$.

3.3.1 Regularizer $\Omega(M)$

We introduce here a regularizer which favors matrices M close to a multiple of the identity matrix. Our objective function (4.4) is not scale-invariant as the threshold to decide whether two feature vectors are similar is set to 1. Therefore, multiplying the matrix by a factor should not be penalized. One could suggest to define the regularizer to be $||s \times M - I||_F$ where s is a free variable introduced to accommodate for the scale of M and to optimize M and s jointly during training. Unfortunately, this regularizer is not convex. Our proposal is to use $\Omega(M) = ||M - sI||_F$. The two formulations are not equivalent. $\Omega(M)$ is scale invariant for diagonal matrices only. But in practice, as the goal of the regularizer is to favor matrices close the identity matrix and therefore diagonal, $\Omega(M)$ do not show a strong bias toward matrices with small coefficients as $||M||_F$ does.

It is possible to avoid dealing explicitly with the additional variable s. Indeed, for any matrix M there is a closed-form solution for s which minimizes the regularizer. It is obtained by solving the equation $\partial||M - sI||_F/\partial s = 0$ which leads to $s = \mathrm{Tr}(M)/n$ where n is the dimensionality of the feature space. The variable s can then be replaced by its expression to obtain a final formulation of our regularizer: $\Omega(M) = ||M - \frac{\mathrm{Tr}(M)}{n}I||_F$.

$\Omega(M)$ is convex as shown by the following proof: $\forall M, M' \in \mathcal{S}_+^n$,

$$\Omega\left(\frac{M+M'}{2}\right) = \left\|\frac{M+M'}{2} - \frac{\mathrm{Tr}(M+M')}{2n}I\right\| \tag{3.13}$$

$$= \frac{1}{2}\left\|M - \frac{\mathrm{Tr}(M)}{n}I + M' - \frac{\mathrm{Tr}(M')}{n}I\right\| \tag{3.14}$$

$$\leq \frac{1}{2}\left\|M - \frac{\mathrm{Tr}(M)}{n}I\right\| + \frac{1}{2}\left\|M' - \frac{\mathrm{Tr}(M')}{n}I\right\| \tag{3.15}$$

$$\leq \frac{\Omega(M) + \Omega(M')}{2}. \tag{3.16}$$

The regularizer $\Omega(M)$ can easily be integrated into any gradient-based optimization scheme as its gradient has a simple closed-form:

$$\frac{\partial\Omega(M)}{\partial M} = \frac{M - \frac{\mathrm{Tr}(M)}{n}I}{\Omega(M)}. \tag{3.17}$$

The regularizer $||M||_F$ favors matrices with small coefficients. When the parameter λ which tunes the regularization strength is large, the use of $||M||_F$ tends to make all distances small and hence changes significantly the balance between similar and dissimilar pairs in the loss function. In extreme cases, only dissimilar pairs' losses are active and similar pairs are completely ignored.

The Regularizer $\Omega(M)$ does not have this issue. It favors matrices which are close to any multiple of the identity but is not biased toward matrices with small diagonal elements. To get an insight about this, let's study the gradient of Ω when M is a 2-by-2 matrix:

$$M = \begin{bmatrix} a & c \\ c & b \end{bmatrix}. \tag{3.18}$$

The gradients with respect to a, b and c are respectively equal to $a-b$, $b-a$ and $4c$.

Table 3.2: Regularizer Comparison on Public Datasets. Performances indicated are Classification Accuracy for MNIST and Reuters, Accuracy for LFW and FNR at FPR=0.1% for FRGC.

	MNIST	Reuters	LFW	FRGC
$\|\|M\|\|_F$	97.77%	88.64%	**87.77%**	1.26%
$\Omega(M)$	**97.86%**	**88.75%**	87.60%	**1.17%**
Relative error improvement	-4.0%	-1.0%	+1.4%	-7.1%

When minimizing $\Omega(M)$ with gradient descent, we easily see that, at the optimum, the off-diagonal element c will have a small absolute value and the diagonal elements a and b are both going to be close to the average of their initial value. Using $\Omega(M)$ as a regularizer for metric learning will not give more weight to the dissimilar pairs than to the similar ones even if the hyper-parameter λ is large.

The issue described above occurs frequently when a metric is learned to adapt features which have been learned for a specific dataset to a new one from a different domain which has few samples for training. We show the benefit of using $\Omega(M)$ rather than Frobenius norm regularizer in a domain adaptation context in Section 3.3.2.

3.3.2 Effect of the Regularizer $\Omega(M)$

In Section 3.3, we presented a new regularizer for metric learning $\Omega(M)$. We compare it to the most common metric learning regularizer, namely the Frobenius norm.

The performances reported in Table 3.2 correspond to the classification accuracy on MNIST and Reuters, accuracy on LFW and False Negative Rate at 0.1% of False Positive Rate on FRGC. $\Omega(M)$ improves performances on 3 out of 4 public datasets. The improvement is most significant on FRGC. This might be linked to the fact that the preprocessing stage described in Section 3.1.3 works particularly well on this dataset and hence constraining the matrix M to be close to the identity is effective.

We explained in Section 3.3 that $\Omega(M)$ has an advantage over the Frobenius norm when the regularization strength hyper-parameter λ is large. Indeed, using $||M||_F$ leads to giving more weight to the dissimilar pairs than the similar ones but our proposed regularizer does not. We illustrate this by performing the following face verification experiment. We use feature vectors produced by a Convolutional Neural Network (CNN)

Table 3.3: Metric Learning for face verification with identity document photos. Comparison of the False Negative Rate at 0.1% of False Negative Rate.

	DB1	DB2	DB3	DB4	DB5	DB6
No Metric Learning	3.1%	4.7%	4.3%	4.0%	5.8%	17.3%
ML with $\|\|M\|\|_F$	3.3%	4.5%	4.4%	1.3%	**5.4%**	16.9%
ML with $\Omega(M)$	**2.8%**	**4.0%**	**3.2%**	**1.2%**	5.5%	**16.2%**
Relative error improvement	-15.1%	-11.1%	-27.3%	-7.7%	+1.9%	-4.1%

trained on 500 000 images of actors downloaded from the Internet [39] and learn a metric to adapt those features to the kind of pictures used in identity documents.

For each regularizer, we learn a metric on 500 images and then evaluate it on 6 proprietary datasets made of photos from passports and other identity documents. Because of the few samples used for training, strong regularization is applied to prevent over-fitting. The different test datasets mainly differ by the type of acquisition process (use of a digital camera, old document scans etc.). We see in Table 3.3 that using Metric Learning improves performance on most of the datasets despite the small number of samples used for training. $\Omega(M)$ significantly outperforms the Frobenius norm on all but one dataset. The performance gap is larger in this experiment than on the public datasets; this is linked to the fact that the hyper-parameter λ (selected by cross-validation) is much greater in this experiment.

3.4 Optimization

3.4.1 Stochastic Gradient Descent

Gradient Descent, also called Steepest Descent, is a standard and generic minimization technique. It starts with an initial solution and iteratively improves it by making a step in the opposite direction of the gradient of the function. Its most basic version uses a fixed step size which needs to be carefully tuned but various strategies have been proposed to speed-up the convergence.

Once the direction is chosen, finding the good step size is an optimization problem of its own called *line search*. Many methods exist, one can use derivative-based methods (1st and 2nd order), Golden Section Search [40] or approximate methods such as

Backtracking Line Search which have been proven to be very efficient. A simpler possibility is to have a predefined sequence of step size ε_t with t being the iteration index. For example, it is common to use $\varepsilon_t = a/(b+t)$ where a and b are hyper-parameters controlling the decrease rate.

The direction of the gradient is not the optimal one. For example, when the function to optimize has narrow valleys, it might lead to strong oscillations from one side of the valley to the other. Newton or Quasi-Newton methods such as L-BFGS [41] use second order derivative information to get a better direction by locally approximating the function by a quadratic form. Conjugate Gradient [42] is also very popular. It imposes orthogonality constraints on subsequent search directions to prevent oscillations.

The methods described above aim at reducing the number of iterations to reach the optimum at the expense of higher computational cost per iteration. The opposite approach can also be considered: doing more iterations but reducing the time taken by each. In machine learning with large scale training sets, computing the gradient is long because of the loop over all the training examples. Stochastic Gradient Descent (SGD) approximates the average gradient over the whole training set by the gradient depending on only one random sample. The resulting approximation is generally too noisy to be used with advanced methods such as Conjugate Gradient and Quasi-Newton methods but it is accurate enough for Steepest Descent. In practice, we use a mini-batch version of SGD. Instead of approximating the gradient by looking at only one example, it uses a small batch of examples randomly selected for each iteration. While the objective functions presented in this chapter are convex and have no local minima, it is not always the case. Another advantage of replacing the exact gradient by a noisy stochastic approximation is that the method has a chance of escaping local minima [43].

Figure 3.6 compares the time of convergence of Gradient Descent and mini-batch SGD with different batch sizes on a toy metric learning problem. In this experiment, we use a fixed step size. We clearly see that SGD is faster than Gradient Descent of several orders of magnitude. On this problem, the best convergence rate is obtained with a batch size of 1000 because it offers a good trade-off between the accuracy of the gradient and its computation time. Moreover, we see that, with a batch size of 10, the gradient approximation is too noisy to allow good convergence and the process does not go below 0.2. It would be necessary to use a smaller step size to get better results with a

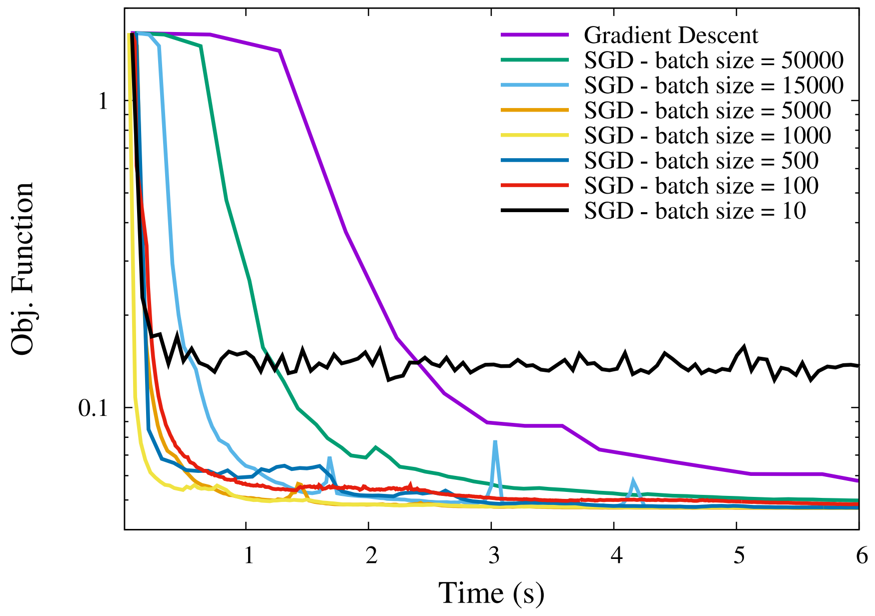


Figure 3.6: Speed of convergence of Gradient Descent and SGD function of the batch size.

small batch size. However a smaller step size would decrease the speed of convergence at the beginning of the optimization process. The best solution is therefore to change the step size during the optimization.

Performing a line search using the entire dataset after each iteration to compute the step size would be very slow. Therefore, most SGD implementations use a predefined step size sequence such as $\varepsilon_t = a/(b+t)$. Under mild assumptions over the objective function and the step size sequence, SGD converges almost surely. With strictly convex functions, such as those presented in this chapter, the expected error decreases in $\mathcal{O}(t^{-1})$ [44].

The hyper-parameters a and b have a significant impact on the convergence speed in practice. To overcome this drawback, we use the Bold Driver method [45] to adapt the step size ε in our implementation of SGD (see Algorithm 1). The step size is initialized with a rather large value ε_0 at the beginning, then, it is updated every p iterations depending on the value of the objective function computed on a validation set $\mathcal{V}$. The step size is divided by 3 if the error has increased since the previous check, or multiplied by 1.1 otherwise. The algorithm stops when the step size falls below a predefined value ε_{end}.

Algorithm 1: Stochastic Gradient Descent

```
Input: Training set $\mathcal{T}$, objective function $\phi$, meta-parameters $p$, $\varepsilon_0$, and $\varepsilon_{end}$
Output: Matrix $M$

/* Initialization                                        */
$\varepsilon \longleftarrow \varepsilon_0$                                  /* Step size */
$\mathcal{V} \longleftarrow$ random subset of $\mathcal{T}$                 /* Validation set */
$E \longleftarrow \infty$                                   /* Training error on $\mathcal{V}$ */
$M \longleftarrow$ random matrix
$M_{prev} \longleftarrow$ M
$i \longleftarrow 1$                                        /* Iteration counter */

/* Optimization                                          */
while $\varepsilon > \varepsilon_{end}$ do
    if $i$ mod $p = 0$ then
        if $E > \phi_{\mathcal{V}}(M)$ then
            $E \longleftarrow \phi_{\mathcal{V}}(M)$
            $M_{prev} \longleftarrow M$
            $\varepsilon \longleftarrow 1.1 \times \varepsilon$
        else
            $M \longleftarrow M_{prev}$
            $\varepsilon \longleftarrow 0.33 \times \varepsilon$
    $i \longleftarrow i+1$
    $\mathcal{T}' \longleftarrow$ random subset of $\mathcal{T}$
    $M \longleftarrow M - \varepsilon \times \frac{\partial \phi_{\mathcal{T}'}(M)}{\partial M}$
```

3.4.2 Bypassing with the Positive-Definiteness Constraint

In metric learning, the matrix M is usually constrained to be positive-semidefinite so that the similarity function is a pseudo-distance. If M has negative eigenvalues, it implies that, for some directions in space, the more different x_i and x_j are, the more similar they would be considered to be. This would clearly be an odd behavior and therefore should be avoided. Gradient-based methods such as those described in Section 3.4.1 do not enforce the constraint and can therefore lead to non valid solutions. Projected Gradient Descent is a standard way to overcome this issue. After each update, the parameter is projected onto the set of feasible solutions. To achieve this, we could add a last line in the while-loop in Algorithm 1:

$$M \longleftarrow \underset{M' \in \mathbb{S}_+^n}{\operatorname{argmin}} \|M - M'\|_F. \tag{3.19}$$

This optimization problem is easy to solve. First, we compute the eigendecomposition of M: $M = UDU^\top$, second, we set all the negative eigenvalues to 0 and, finally, we reconstruct the matrix M. However, the projection is a major bottleneck in making the optimization process efficient because eigendecomposition has a complexity of $\mathcal{O}(n^3)$.

We prefer another solution to deal with the positive-definiteness constraint of M which consists in making a change of variable $M = L^\top L$ where $L \in \mathbb{R}^{n\times n}$ and performing an unconstrained optimization with respect to L. As we shall see, despite the fact that some objective functions are convex in M but not in L, there is no risk to get stuck in a non-global minimum. The following proof has been proposed by Maurer in [46]. We call $\phi(M)$ our objective function which is continuous and does not have any non local minimum. We define the function ψ by $\psi(L) = \phi(L^\top L)$. Let us assume that $\psi(\widehat{L})$ is a minimum and $\phi(\widehat{L}^\top \widehat{L})$ is not. We can build a sequence of matrices $M_n \in \mathbb{S}^n_+$ such that $M_n \xrightarrow{n\to\infty} \widehat{L}^\top \widehat{L}$ and $\forall n,\ \phi(M_n) < \phi(\widehat{L}^\top \widehat{L})$. The polar decomposition of L can be written as $\widehat{L} = U\sqrt{\widehat{L}^\top \widehat{L}}$ where U is a unitary matrix. We can build a sequence of matrices $L_n = U\sqrt{M_n}$ which converges to $U\sqrt{\widehat{L}^\top \widehat{L}}$. We then have the following relations $\psi(L_n) = \phi(L_n^\top L_n) = \phi(M_n) < \psi(\widehat{L})$ which are in contradiction with the assumption that $\widehat{L}$ is a minimum of ψ. Therefore, all minima of ψ are global and have the same value as the minimum of ϕ.

Chapter 4

Large Margin Local Metric Learning

In machine learning, linear methods are appealing because they are relatively simple and often powerful enough to deliver good performance. When this is not the case, one way to go beyond linear functions is to use kernels. However, it leads to methods which do not scale well in terms of speed when the number of training examples grows. An alternative way to extend linear methods which does not suffer this limitation is to use local functions. This scheme has been used for classification by Ladický and Torr in [47]. Their classifier is a weighted combination of several linear SVMs, the weights of the combination depending on where the point to classify is located.

This idea can also be applied to metric learning. In Section 1.3, we have introduced this concept and briefly described the state-of-the-art methods. Previous methods suffer from two main limitations: they cannot deal with an arbitrary large number of classes and/or they are designed solely for nearest neighbor classification and, therefore, cannot work on pairs of points which none of them are in the training set. In this chapter, we introduce a flexible local metric learning method called Large Margin Local Metric Learning (LMLML) which overcomes these limitations. It computes a set of local metrics which are combined into an adaptive similarity function with the help of a soft partitioning of the feature space. The optimization of the local metrics is formulated as a convex problem which favors a large margin solution. Our experiments show that LMLML outperforms or matches state-of-the-art results on various datasets. We have originally presented LMLML in a paper published at ECCV14 [48].

This Chapter describes the proposed method. We start by detailing LMLML's model and each step of its optimization and, then, demonstrate its effectiveness on various datasets in Section 4.3

4.1 Local Metric

We start by applying the preprocessing step described in Section 3.1.3 to the raw feature vectors extracted from the data. Whenever we mention a feature vector x in the reminder of this chapter, it refers to the n-dimensional preprocessed and normalized vector. Let $\mathcal{S}_+^n$ be the set of $n \times n$ PSD matrices. The usual squared Mahalanobis distance associated with a matrix $M \in \mathcal{S}_+^n$ and evaluated on a pair of data points $(x_i, x_j) \in \mathbb{R}^n \times \mathbb{R}^n$ is given by $(x_i - x_j)^\top M (x_i - x_j)$. In LMLML, the matrix M is replaced by a matrix-valued function $\mathcal{M}_\theta : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathcal{S}_+^n$ which is defined, for every $(x_i, x_j) \in \mathbb{R}^n \times \mathbb{R}^n$, as a convex combination a $K+1$ matrices

$$\mathcal{M}_\theta(x_i, x_j) = \sum_{k=0}^{K} w_\theta^k(x_i, x_j) M_k \tag{4.1}$$

where $w_\theta^k(x_i, x_j)$ are nonnegative weights which will be defined below. The resulting similarity function is given, for every $(x_i, x_j) \in \mathbb{R}^n \times \mathbb{R}^n$ by the formula

$$d^2(x_i, x_j, \mathcal{M}_\theta) = (x_i - x_j)^\top \mathcal{M}_\theta(x_i, x_j)(x_i - x_j). \tag{4.2}$$

The smoothness of the matrix-valued function $\mathcal{M}_\theta$ is a desirable property because it ensures the similarity function be local. It also prevents abrupt changes which, as we observed in our experiments, degrade performance. In order to favor the smoothness, we use weight functions w_θ^k which vary smoothly across the input space. As we want the similarity function to be *local*, it makes sense to use a soft partitioning of the input space to compute the weights $w_\theta^k(x_i, x_j)$. To this end, we employ a Gaussian Mixture Model (GMM) with K components of parameter $\theta = \{\alpha_k, \mu_k, S_k\}_{k=1\ldots K}$, where α_k is the mixing probability of the Gaussian with mean μ_k and covariance matrix S_k. The weights are defined by the formula:

$$w_\theta^k(x_i, x_j) = \begin{cases} \beta & \text{if } k = 0 \\ P(k|x_i, \theta) + P(k|x_j, \theta) & \text{otherwise} \end{cases} \tag{4.3}$$

where β is a positive constant and $P(k|x, \theta)$ is the posterior probability that the point x has been generated by the Gaussian k of the GMM. Notice that for every $(x_i, x_j) \in$

$\mathbb{R}^n \times \mathbb{R}^n$, we always have $\sum_{k=0}^{K} w_\theta^k(x_i,x_j) = 2+\beta$. The GMM parameter is initialized using the standard Maximum Likelihood EM algorithm and refined during the metric optimization process.

Note that each local metric has a strong influence only within a specific region, that is M_k has a large weight in $\mathcal{M}_\theta(x_i,x_j)$ if x_i or x_j is strongly associated with Gaussian k and even more so if both are. In the face verification example mentioned in the introduction, the soft partitioning tends to roughly regroup faces by gender and ethnicity (see Section 4.3.3). Thus $\mathcal{M}_\theta(x_i,x_j)$ emphasizes the features which are the most discriminative to compare people with similar gender/ethnicity than x_i and/or x_j.

The metric M_0 is associated with a constant weight and is therefore a global metric, it handles the part of the similarity function which is common to the whole input space. The metrics M_k with $1 \leq k \leq K$ deal with the local adaptations of the similarity function over the regions defined by the Gaussians of the GMM. Our model is a generalization of global metric and purely local metric as the parameter β allows to balance the influence of the global metric M_0 and the local metrics M_k in the matrix $\mathcal{M}_\theta(x_i,x_j)$. The larger K is, the more the model will be able to handle subtle local adaptations. Among the values of K obtaining comparable performance during cross-validation, the smallest should be preferred because speed and memory occupancy for training and testing grow linearly with K. The impact of K on the performance is studied in Section 4.3.2. If $K = 0$, $K = 1$ or $\beta \to \infty$ our model is equivalent to a global linear metric.

One could think that adding the matrix M_0 in our model is unnecessary because the same similarity function can be written without M_0 by integrating it into the other matrices M_k. However, with the help of an appropriate regularizer (see Section 4.1.1), the actual formulation allows one to use all the training points to learn the part of the similarity function which is common to the whole space and this leads to better generalization performance. We experimentally verified that when the global metric is removed, performance deteriorates significantly.

4.1.1 Objective Function

Let $\mathcal{T} = \{(i,j)\}$ denote the index set of training pairs and let r_{ij} be a label which is equal to 1 if (x_i,x_j) is a similar pair and -1 otherwise. The objective function of LMLML is given by

$$\phi(\mathbf{M},\theta) = \frac{1}{|\mathcal{T}|} \sum_{(i,j)\in\mathcal{T}} \ell_\gamma\left(r_{ij}, d^2(x_i,x_j,\mathcal{M}_\theta)\right) + \lambda \sum_{k=0}^{K} \Omega\left(M_k\right) \tag{4.4}$$

where $\mathbf{M} = \{M_0,\ldots,M_K\}$ and the loss function ℓ_γ is defined by Equation (3.6). The empirical loss is high for similar pairs which have large distances and dissimilar pairs with small distances (see Section 3.1.2 for more details about this loss function).

The function Ω is the convex regularizer proposed in Section 3.3. Other standard matrix regularizers such as the Frobenius norm or the trace norm could also be used but we obtained better results with Ω. The overall expression $\sum_{k=0}^{K}\Omega\left(M_k\right)$ is a mixed-norm regularizer. It favors solutions where the part of the similarity function common to the whole space is concentrated in the matrix M_0. λ tunes the strength of the regularization, it needs to be set by cross-validation. We seek to minimize $\phi(\mathbf{M},\theta)$ with respect to $\mathbf{M}$ and θ under the constraint that $M_k \in \mathcal{S}_+^n$, for every $k \in \{0,\ldots,K\}$.

The function $\mathbf{M} \mapsto \phi(\mathbf{M},\theta)$ is convex in $\mathbf{M}$ for every choice of the parameter vector θ, because it is a sum of convex functions. Indeed the regularizer Ω in (4.4) is convex (see Section 3.3.1) and the empirical loss terms are also convex. Specifically, by equations (4.1) and (4.2), we see that the quantity $d^2(x_i,x_j,\mathcal{M}_\theta)$ is linear with respect to each of the matrices M_k. Since the hinge loss is convex and the composition of a convex function with a linear mapping is convex (see [49], 3.2.2), we conclude that the function $\phi(\mathbf{M},\theta)$ is convex in $\mathbf{M}$.

The positive-definiteness constraints on the matrices M_k could be enforced by using a projected gradient descent. However, this would be rather slow as it requires performing an eigen-decomposition of each matrix M_k to project it on the positive-definite cone after each iteration. To overcome this difficulty, inspired by previous work in [15], we transform the constrained problem into an unconstrained one by making the change of variable and $M_k = L_k^\top L_k$, for $k = 0,1\ldots,K$. We introduce the objective function ψ as

$$\psi(\mathbf{L},\theta) = \phi(\mathbf{M}(\mathbf{L}),\theta) \tag{4.5}$$

where $\mathbf{L} = \{L_0,\ldots,L_K\}$ and $\mathbf{M}(\mathbf{L}) = (L_0^\top L_0, L_1^\top L_1,\ldots,L_K^\top L_K)$. Solving this optimization problem does not create any local minima issue. The proof given in Section 3.4.2

is trivial to extended to the multi matrix case and therefore, as ϕ is convex in $\mathbf{M}$, we can minimize the unconstrained function ψ without risking getting stuck in a non optimal local minimum. Notice that this reasoning only applies because the matrices L_k are square. Using rectangular matrices would be equivalent to adding rank constraints on the matrices M_k and therefore making the initial optimization problem non convex.

4.1.2 Alternate Minimization Scheme

We optimize $\psi(\mathbf{L}, \theta)$ with an alternate minimization scheme based on gradient descent. We first initialize the GMM using standard Maximum Likelihood EM algorithm and compute the weights $w_\theta^k(x_i,x_j)$ using (4.3). We then interleave gradient steps with respect to $\mathbf{L}$ and gradients steps with respect to θ until the loss improvement between two optimization steps of $\mathbf{L}$ is smaller than a predefined threshold. Optimization is performed by mini-batch stochastic gradient descent (SGD) with a adaptive step size decrease (see Section 3.4.1). For each gradient step, only a random subset of the training samples is used. Let $\psi_{\mathcal{T}'}(\mathbf{L}, \theta)$ be the approximation of the objective function using only the batch $\mathcal{T}' \subset \mathcal{T}$, its gradient with respect to L_k is equal to

$$\frac{1}{|\mathcal{T}'|} \sum_{(i,j)\in\mathcal{T}'} \frac{\partial \ell_\gamma\left(r_{ij}, d^2(x_i,x_j,\mathcal{M}_\theta)\right)}{\partial L_k} + \lambda \frac{\partial \Omega\left(L_k^\top L_k\right)}{\partial L_k}$$

with

$$\frac{\partial \ell_\gamma\left(r_{ij}, d^2(x_i,x_j,\mathcal{M}_\theta)\right)}{\partial L_k} = \begin{cases} 0 \text{ if } r_{ij}(1-d^2(x_i,x_j,\mathcal{M}_\theta)) > \gamma \\ \\ \dfrac{2r_{ij}w_\theta^k(x_i,x_j)L_k(x_i-x_j)(x_i-x_j)^\top}{\gamma} \text{ otherwise} \end{cases} . \quad (4.6)$$

The gradient of the regularizer is given by (3.17). Note that in LMLML, the matrices L_k are jointly optimized: they all impact the value of $\mathcal{M}_\theta(x_i,x_j)$ and therefore the gradient with respect to each L_k.

Next, we detail the formula of the gradient of $\psi_{\mathcal{T}'}(\mathbf{L}, \theta)$ with respect to μ_k, S_k and α_k. To this end, we first define the auxiliary functions

$$g_{k,k'}(x) = \begin{cases} -\sum_{k''\neq k} \alpha_{k''} P(x|k'') & \text{if } k = k' \\ \\ \alpha_{k'} P(x|k') & \text{otherwise} \end{cases} . \quad (4.7)$$

The gradient with respect to μ_k is equal to

$$\frac{1}{|\mathcal{T}'|}\sum_{(i,j)\in\mathcal{T}'} r_{ij}\sum_{k'}\|L_k(x_i-x_j)\|^2\left(f_k^{\mu}(x_i)g_{k,k'}(x_i)+f_k^{\mu}(x_j)g_{k,k'}(x_j)\right) \tag{4.8}$$

where

$$f_k^{\mu}(x)=\frac{\alpha_k P(x|k)}{\left(\sum_{k'}\alpha_{k'}P(x|k')\right)^2}S_k^{-1}(\mu_k-x). \tag{4.9}$$

The gradient with respect to S_k is equal to

$$\frac{1}{|\mathcal{T}'|}\sum_{(i,j)\in\mathcal{T}'} r_{ij}\sum_{k'}\|L_k(x_i-x_j)\|^2\left(f_k^{S}(x_i)g_{k,k'}(x_i)+f_k^{S}(x_j)g_{k,k'}(x_j)\right) \tag{4.10}$$

where

$$f_k^{S}(x)=\frac{\alpha_k P(x|k)}{2\left(\sum_{k'}\alpha_{k'}P(x|k')\right)^2}\left(S_k^{-1}-S_k^{-1}(\mu_k-x)(\mu_k-x)^\top S_k^{-1}\right). \tag{4.11}$$

Finally, the gradient with respect to α_k is equal to

$$\frac{1}{|\mathcal{T}'|}\sum_{(i,j)\in\mathcal{T}'} r_{ij}\sum_{k'}\|L_k(x_i-x_j)\|^2\left(f_k^{\alpha}(x_i)g_{k,k'}(x_i)+f_k^{\alpha}(x_j)g_{k,k'}(x_j)\right) \tag{4.12}$$

where

$$f_k^{\alpha}(x)=-\frac{P(x|k)}{\left(\sum_{k'}\alpha_{k'}P(x|k')\right)^2}. \tag{4.13}$$

4.2 Computing the GMM on a Low Dimensional Embedding

The weights $w_\theta^k(x_i,x_j)$ presented in Section 4.1 are based on the soft partitioning of the space derived from the GMM of parameter θ. Instead of training the GMM on the original features, we propose to first project the data into a very low dimensional discriminative embedding. We use 10 dimensions in all our experiments, much less than the several hundreds of dimensions kept after the PCA used in the preprocessing step. This serves two purposes: first, the low dimensionality of the embedding favors the smoothness of weight functions. Second, for local metric to be effective it is needed that points which are hard to discriminate using a global metric have similar weights so that the local metrics focus on the locally discriminative information.

The very low dimensionality inherently limits the amount of information preserved. While enough information is preserved to perform a soft partitioning of the data, the actual similarity should rather be computed in a higher dimensional feature space. The low dimensional embedding presented in this section is only used to compute the GMM parameter and derive the weights $w_\theta^k(x_i, x_j)$ but the metric is computed directly on the feature vectors output by the preprocessing stage (i.e. the matrices M_k are of order n).

PCA should not be used to reduce the dimensionality to 10 dimensions because it would lead to a massive loss of discriminative information. We propose to use a method inspired by low rank global metric learning methods [50] to obtain a transformation matrix $V \in \mathbb{R}^{m \times n}$ which embeds the data into a low dimensional discriminative feature space of size m. We find the transformation V by minimizing an objective function similar to (4.4):

$$\frac{1}{|\mathcal{T}|} \sum_{(i,j) \in \mathcal{T}} \ell_\gamma\left(r_{ij}, \|V(x_i - x_j)\|^2\right) + \lambda \Omega(V^\top V). \tag{4.14}$$

This optimization problem is not convex, therefore, there is no guarantee to find the global minimum In practice we observed that the initialization has only a limited impact on the final performance. The minimization is performed by stochastic mini-batch gradient descent.

To illustrate the impact of the dimensionality on the smoothness of the weight functions, we compute low dimensional embeddings for several dimensionality values m on MNIST. For each value we perform the following operations:

1. Estimation of the GMM parameters (for this experiment we have arbitrarily chosen $K = 5$),
2. Computation of the posterior probability distribution $P(\cdot|x)$ for every x in the training set,
3. Computation of the Bhattacharyya distance

$$d_B\left(P(\cdot|x_i), P(\cdot|x_j)\right) = -\log \sum_k \sqrt{P(k|x_i)P(k|x_j)} \tag{4.15}$$

between the distribution associated with each x and those of its 3 nearest neigh-

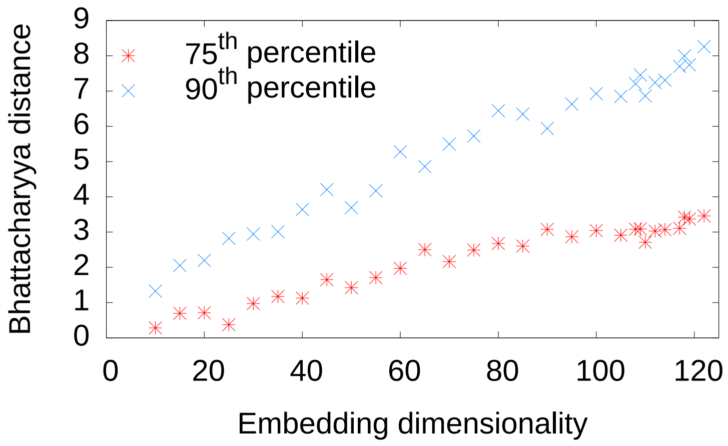


Figure 4.1: Percentiles of Bhattacharyya distance between neighbors function of the embedding dimensionality m.

bors.

Figure 4.1 shows the 75th and 90th percentiles of the computed Bhattacharyya distance function of the dimensionality of the embedding. We can see that a larger embedding dimensionality leads to a larger Bhattacharyya distance between nearby points which means less smooth weight functions w_θ^k. This smoothness matters: on high dimensional data such as that used for face verification (see Section 4.3.3), we have observed that more than half of the benefit of using local metric vanishes if the GMM is computed in the original feature space. This supports the claim that the GMM should be computed on a low dimensional embedding.

4.3 Experiments

LMLML's performance is assessed through a set of extensive experiments. First, we illustrate the advantage of local metric learning on synthetic datasets. Second, we compare LMLML to other metric learning methods on a large variety of datasets which are presented in Chapter 2.

4.3.1 Synthetic Dataset

Using a single metric to compare two feature vectors may often be too restrictive to obtain good performance. Local metric learning overcomes this limitation by allowing adaptation of the similarity function to the regions of the input space the two feature vectors belong to. To illustrate this advantage we build a 2D synthetic dataset using the following procedure.

A GMM with 3 Gaussians with significant overlap is used to create synthetic datasets. We generate a similar pair in 3 steps. First, we draw the pair center using the GMM. Second, we draw the orientation of the pair in a Gaussian distribution whose mean depends on the component of the GMM the pair center belongs to. Third, the distance between the two vectors of the pair is also drawn from a Gaussian distribution and the two point locations are computed. The process to obtain a similar pair (x_1, x_2) is summarized below:

Step 1:

$$z \sim Categorical(\{1/3, 1/3, 1/3\}) \tag{4.16}$$

$$c \sim \mathcal{N}(\mu_z^c, 0.3 \times I) \tag{4.17}$$

Step 2:

$$o \sim \mathcal{N}(\mu_z^o, 15) \tag{4.18}$$

Step 3:

$$m \sim \mathcal{N}(0, 0.2) \tag{4.19}$$

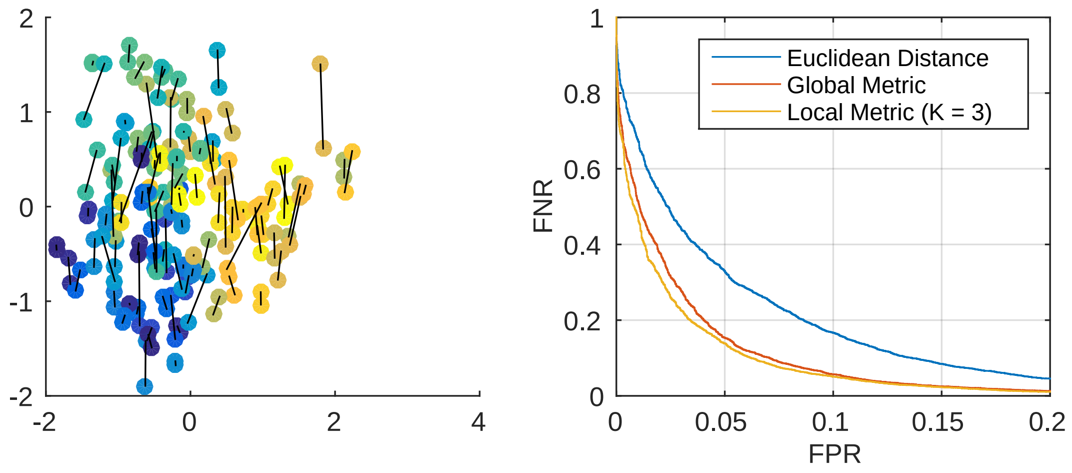
$$x_1 = c + rot\left([0, m]^\top, o\right) \tag{4.20}$$

$$x_2 = c - rot\left([0, m]^\top, o\right) \tag{4.21}$$

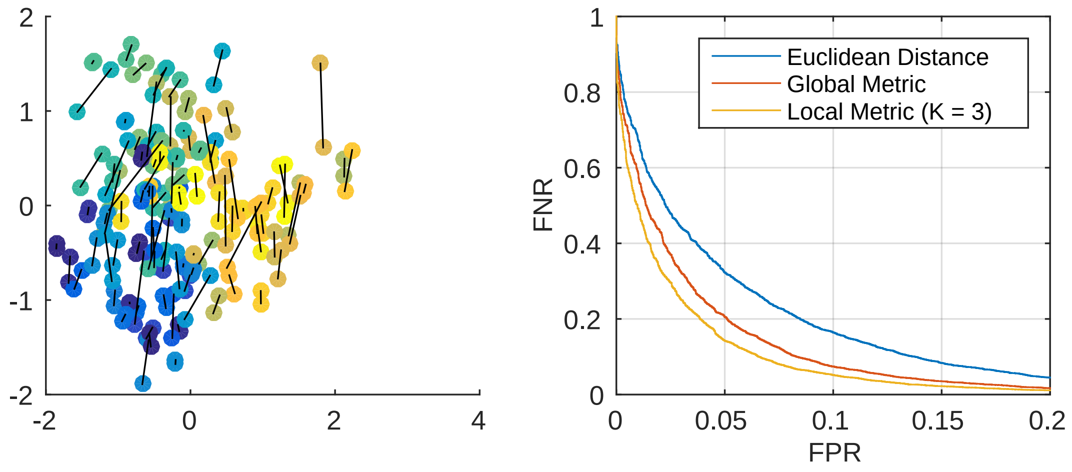
where $\mu_1^c = [-1, -1]^\top$, $\mu_2^c = [-1, 1]^\top$, $\mu_3^c = [1, 0]^\top$, $\mu_{\{1,2,3\}}^o$ are the mean angles associated with each Gaussian and $rot(v, o)$ is the rotation operator which applies a rotation of angle o to the vector v. By varying angles $\mu_{\{1,2,3\}}^o$ used in Step 2, we can tune how much the orientation of similar pairs changes when moving across the input space. The same points are used for dissimilar pairs which are simply made of points which have been generated from different pair centers.

When the orientations of positive pairs belonging to different components are close (see Figure 4.2a), the global metric works well and the local metric only offers a limited improvement. When the change in orientation of the positive pairs becomes more significant, performance of the global metric degrades much faster than that of local metric (Figure 4.2b, 4.3a and 4.3b).

Section 4.1.2 describes LMLML's optimization scheme which alternates between gradient steps with respect to the matrices $M_{0..K}$ and with the GMM parameters θ. Fig-



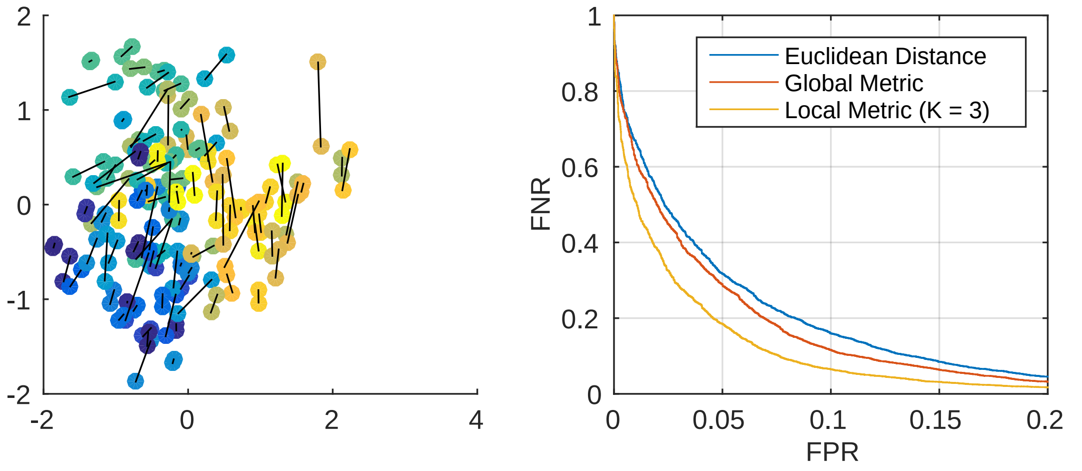
(a) Angle mean difference 5°



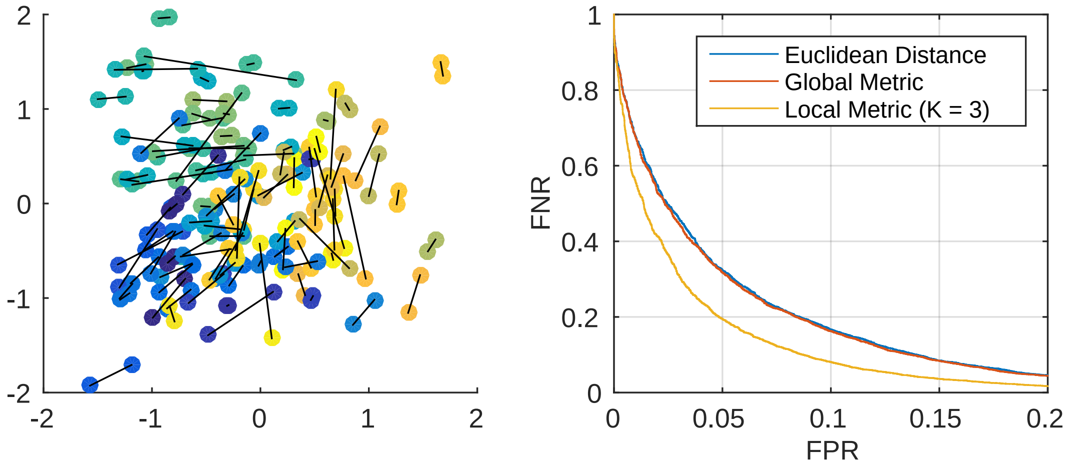
(b) Angle mean difference 15°

Figure 4.2: Synthetic data with small mean angle difference between the 3 GMM components. Left: Examples of positive pairs (points linked by black segments). Right: Performance of the Euclidean Distance, Global Metric and Local Metric.

ure 4.4 shows that performance improves slightly when θ is optimized and not kept fixed at the value output by the EM initialization. We also conducted experiments on the synthetic dataset where θ was randomly initialized. In this case, the performance gap between with and without optimization of the GMM parameters is wider. However, on most real datasets, we observed that optimizing θ had a limited impact on the performance. So, in the experiments described in the rest of this chapter, the optimization of θ has been disabled to speedup the training.



(a) Angle mean difference 30°



(b) Angle mean difference 45°

Figure 4.3: Synthetic data with large mean angle difference between the 3 GMM components. Left: Examples of positive pairs (points linked by black segments). Right: Performance of the Euclidean Distance, Global Metric and Local Metric.

4.3.2 Nearest Neighbor Classification

Metric learning is often use as a mean to improve k-nearest neighbors classification. Indeed, some of the most prominent metric learning methods such as LMNN [7] have been specifically designed for this purpose. We evaluated LMLML on the five classification datasets described in Section 2.2 and compared it to both global and other local metric learning methods. In this section we present classification rates obtained exclusively with 1-nearest neighbor classification. Using more neighbors consistently improves performance of all methods but does not change the observations about the metric learning methods evaluated.

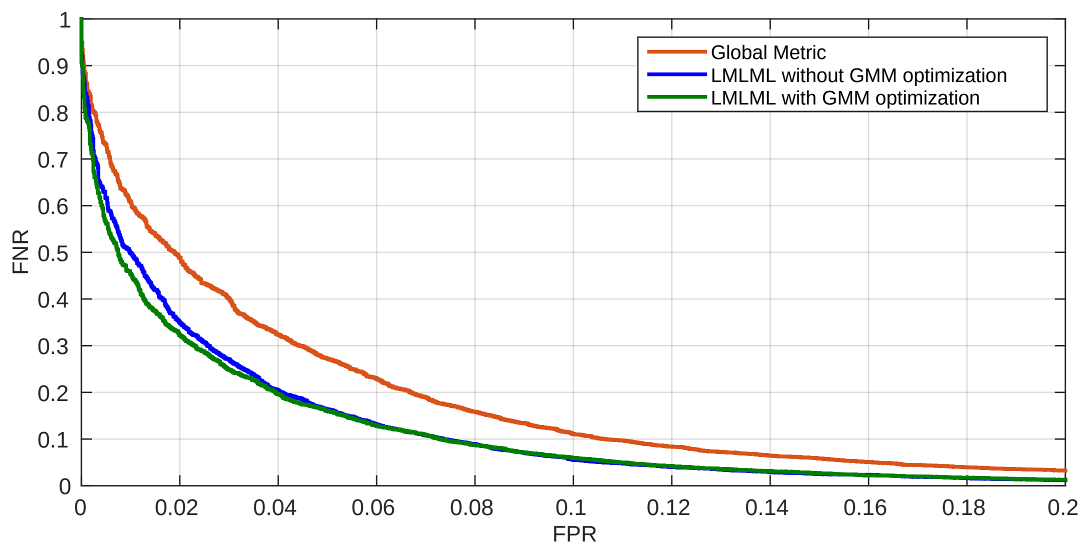


Figure 4.4: Optimizing both the matrices $M_{0..K}$ and the GMM parameters θ improves performance on the synthetic dataset.

Performances of all methods are summarized in Table 4.1. Results of other methods on MNIST, Isolet and Letter are borrowed from [29]. All hyper-parameters have been set by cross-validation. We observe that using a local metric leads to improved performance compared to a global one. The relative decrease in error rate of LMLML when $K > 1$ compared to $K = 0$ ranges from 2% to 24%. The optimal value for K varies from one dataset to another but is generally between 3 and 5. When the risk of over-fitting is small because feature vectors are of low dimensionality and the training set is large, larger values of K can lead to better performance. Figure 4.5 shows LMLML's accuracy on MNIST for all values of K up to 30, the best performance is obtained using $K = 17$. The computation of the GMM parameters is not convex therefore different initializations lead to different performance, the error bars show the performance variability. The classification rate of LMLML on MNIST in Table 4.1 corresponds to $K = 19$ which is the value selected by cross-validation.

LMLML is compared to six others methods designed for nearest neighbor classification: two global metric learning algorithms LMNN [7] and BoostMetric [14], three local metric learning methods MM-LMNN [10], GLML [28] and PLML [29] and a multi-class non linear SVM with one-against-all strategy (the best kernel has been chosen by inner cross-validation). LMLML outperforms the other methods on the five datasets. This good performance can be partly attributed to the use of local metric as

Table 4.1: Classification Rates

	MNIST	Isolet	Letter	20newsgroup	Reuters
LMLML	**98.10%**	**96.02%**	**97.60%**	**77.06%**	**89.03%**
LMLML $K = 0$	97.86%	94.74%	97.21%	76.28%	88.75%
PLML	97.30%	95.25%	97.22%	-	87.39%
LMNN	97.30%	95.51%	96.08%	75.54%	88.87%
MM-LMNN	93.24%	84.61%	95.02%	-	-
Non linear SVM	97.62%	95.19%	96.64%	-	-
GLML	84.02%	84.03%	93.86	-	-

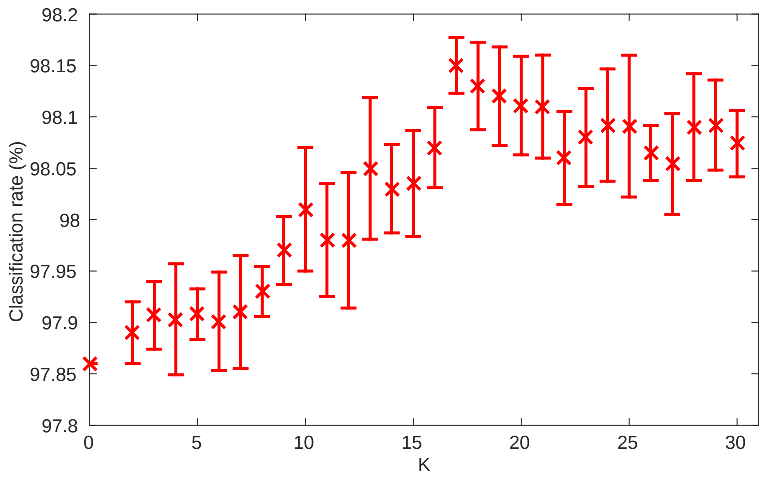


Figure 4.5: Impact of K on LMLML's performance on MNIST.

LMLML with $K = 0$ is beaten by some of the 6 other methods on three datasets out of five.

4.3.3 Face Verification

In nearest neighbors classification, distances are used to rank the items of a gallery database function of their similarity to a specific target. It is not an issue if some targets tend to produce larger or smaller distances than others. However, this is a critical issue in applications where distances are thresholded to decide whether two feature vectors are similar. A good example is face verification which consists in comparing two face images to assess whether both images depict the same person. The identities presented for face verification do not belong to the training dataset. We have performed experiments with two face verification datasets: FRGC and LFW, which are described in Section 2.3.

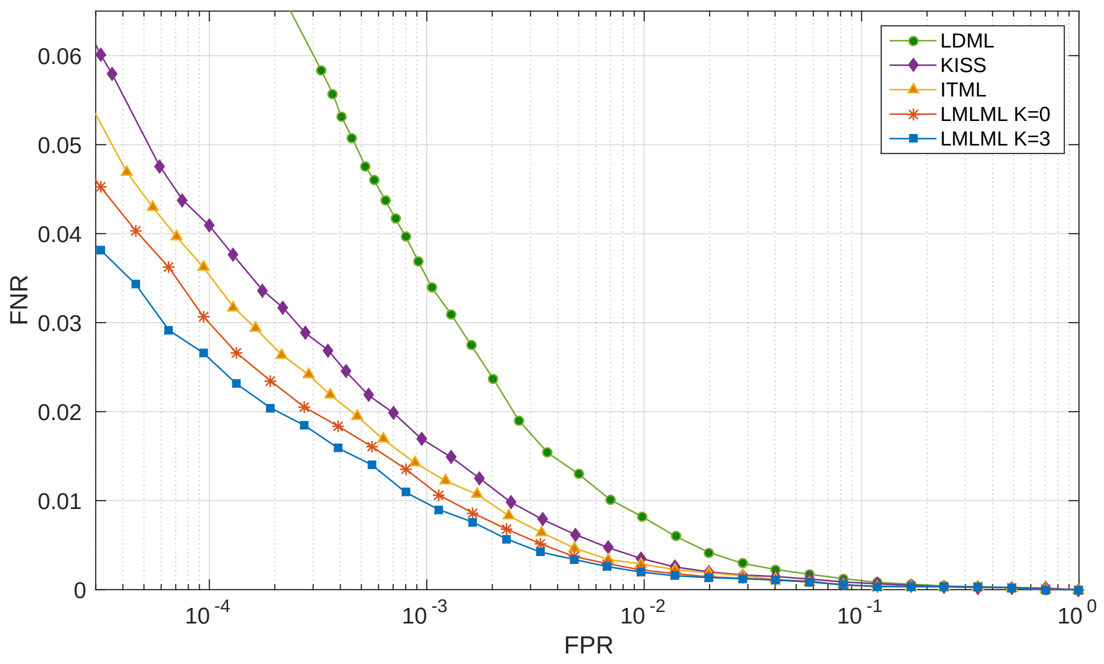


Figure 4.6: Performance of LMLML and other methods on FRGC

4.3.3.1 FRGC

To perform face verification, a method needs to be able to compute a similarity with pairs of never seen points so we cannot compare LMLML to previous local metric learning methods. Figure 4.6 shows the DET curves of LMLML, KISSME [5], ITML [12] and LDML [11]. We used the code available on their respective author's website and cross-validated their parameters in the same way we did with LMLML following the authors' recommendations. Our approach obtains better verification rates at all false positive rates. Moreover, we see that LMLML with $K = 3$ achieves better performance at low false positive rates than with $K = 0$. It means that, as expected, the similarity function's ability to adapt to local specificities leads to a better discriminative power with close-by feature vectors.

In order to gain more insights about the good performance achieved by our method, we observed the distribution of the faces among the different Gaussians of the GMM. Each row of Figure 4.7 shows the five faces with the highest posterior probabilities for each Gaussian. Thanks to the use of our discriminative low dimensional embedding described in Section 4.2, the GMM captures interesting properties of the faces: the first group is mostly populated of Asian people, the second of Caucasian females and the third of Caucasian males. This grouping is totally unsupervised as no

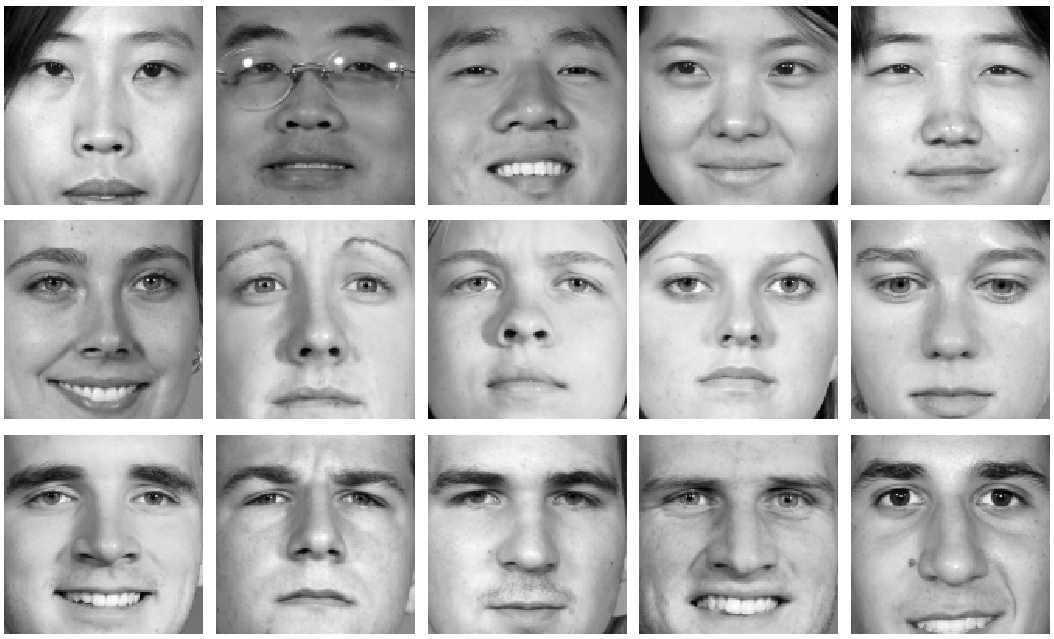


Figure 4.7: Each row shows the images with the highest posterior probability $P(k|x)$ for a specific k.

gender or ethnicity information has been given to the algorithm. It allows LMLML to adapt the similarity function to the specificities of these groups and it contributes to its good performance on FRGC.

4.3.3.2 LFW

Results on LFW in restricted setting (see Section 2.3.1 for more information about this evaluation setting) are shown in Table 4.2. LMLML obtains the 2^{nd} best performance with an accuracy of 87.77%, just behind ECSLDA [51] which achieves 87.97%. However, the parameter selection process consistently chooses $K = 0$ (only a global metric) for all the folds meaning that local metrics does not help on this dataset. Our explanation is that LFW's major challenge is its wide intra-class variability and this issue is not addressed by the use of a local metric. We looked at the results of the soft-partitioning on LFW and saw that images seem to be randomly scattered among the different clusters. Different images of a single person do not tend to have similar values of $P(k|x)$. Local metric does not help on datasets which are too difficult to obtain a meaningful soft-partitioning in the low dimensional embedding described in Section 4.2.

Table 4.2: Accuracy on LFW.

Method	Accuracy	Method	Accuracy
LMLML	87.77% ± 0.55	ITML [12]	79.98% ± 0.39
ECSLDA [51]	**87.97%** ± 0.43	Sub-ITML [18]	83.98% ± 0.48
Sub-SML [18]	86.73% ± 0.53	LDML [11]	80.65% ± 0.47
CSML [16]	85.57% ± 0.52	Sub-LDML [18]	82.27% ± 0.58
KISSME [5]	83.37% ± 0.54	SILD [52]	80.07% ± 1.35
DML-eig [53]	82.28% ± 0.41		

4.4 Conclusion

In this chapter, we have introduced a new local metric learning algorithm. The data is embedded into a discriminative low dimensional space to compute a soft partitioning of the input space. This embedding is then used to define a smooth locally adapting similarity function. Our method overcomes the limitations of previous local metric learning methods, it is as flexible as global metric learning methods and can therefore be applied to wide variety of scenarios. The good performances of LMLML has been demonstrated on seven different datasets.

Chapter 5

Deep Metric Learning

Artificial Neural Networks are methods with a long history in machine learning, the first models, such as the Perceptron [54], go back to the 50s. A neural network defines a parametric mapping function $f_\theta : X \mapsto Y$ where θ denotes its set of parameters of the function. It can be used to perform many machine learning tasks such as classification, regression, dimensionality reduction etc.

Feedforward neural network is a popular class of neural network where the mapping function f_θ is the composition of several functions $f_\theta = g_1 \circ g_2 \circ \cdots \circ g_K$. Each function is either a parametric linear function such as $g_i(y) = W_i y + b_i$ or a non linear function with few or no parameter such as the hyperbolic tangent or the Rectified Linear Unit (ReLU) $g_i(y) = \max(0, y)$. The succession of a linear function and a non linear activation function is commonly referred to as *layer*. The set of parameters θ is the collection of all layers' parameters such as the matrices W_i and the vectors b_i of the linear layers. The resulting function f_θ can be complex and highly non linear. The architecture of a neural network (number of layers and their type, size etc.) is defined specifically for each use case. It depends on the input, the task and the amount of available training data.

The training of an artificial neural network consists in finding the values of the parameters in θ which optimize an objective function. The objective function depends on the task at hand. For classification, the most common approach is to build a network with as many outputs as classes whose goal is to give the highest value to the output corresponding to the correct class. The objective function maximized during training is generally the average softmax cross-entropy between the output vectors and vectors containing a 1 for the correct class and 0 for the others. Other multiclass losses could be used, e.g. the multiclass hinge loss. For regression, the square loss between the output and the ground truth is the most popular choice.

The optimization is usually performed by stochastic gradient descent and hence the objective function needs to be differentiable almost everywhere. The gradient with respect to the parameters of the different layers is iteratively computed from the last layer to the first one using the chain rule for composite functions differentiation. This process is called gradient backpropagation.

The research on neural networks was very active in the 80's but they were progressively replaced by Support Vector Machine and other kernel methods during the 90's. Since 2010, there is a strong renewed interest for neural networks because they recently achieved outstanding performance in many pattern recognition tasks. For example, in computer vision, they became increasingly popular since a Convolutional Neural Network (a feedforward neural network where linear operations are convolutions) won the ImageNet challenge in 2012 by a wide margin [55]. The field has been renamed Deep Learning, the notion of depth being a reference to the large number of layers used in modern neural network architectures. This recent performance surge is due to the combination of three main factors:

1. algorithmic advances such as new non linear activation functions (e.g. ReLU [56]) or new regularization techniques (e.g. dropout [57]),
2. availability of large labeled datasets for training (for example, 1.2 million images for ImageNet),
3. strong increase in computational power with the use of GPUs for general purpose computing.

The two last points are tightly linked as a large amount of data is useless without computational power to use it and vice and versa. GPUs are especially efficient for the type of operations used to train and run neural networks thanks to their massively parallel architecture.

A neural network maps an element from X to a element of Y. To create a neural network-based similarity function, we choose Y to be a discriminative embedding of the input space X and use the similarity function $||f_\theta(x_i) - f_\theta(x_j)||^2$. This is an non-linear extension of the Mahalanobis distance defined by $||Lx_i - Lx_j||^2$ with $L \in \mathbb{R}^{n \times n}$ presented in Section 1.2. By analogy with Linear Metric Learning, we name neural network based-similarity function learning Deep Metric Learning even if the similarity function does not satisfy the triangular inequality.

In the next section, we describe how to use neural networks for similarity function learning. We then evaluate the performance of Deep Metric and compare it to linear metric learning and LMLML presented in Chapter 4 before concluding this chapter.

5.1 Neural Network for Similarity Function Learning

To train a neural network to embed the input data into a discriminative space, we apply the same methodology as that presented in Section 3.1.2 to learn a linear metric. Namely, we optimize the objective function

$$\min_{\theta} \sum_{(i,j)\in\mathcal{T}} \ell_\gamma\left(r_{ij}, \|f_\theta(x_i) - f_\theta(x_j)\|^2\right) + \lambda R(\theta) \tag{5.1}$$

where ℓ_γ is the hinge loss defined by Equation (3.6), $R(\theta)$ is a regularizer and λ allows to tune the intensity of the regularization. This approach is similar to that proposed for example in [58, 25, 26].

The training of the similarity function is composed of two steps. First, a set of pairs (similar and dissimilar) is built using the labels associated with the data. Second, we use stochastic gradient descent to minimize the above objective function. In the next subsections, we describe more in details the architecture of the network and present methods to create good training pairs.

5.1.1 Network details

When input data has a specific structure, it can be taken into account in the design of the network architecture. For example, when inputs are images, it is common in image processing to apply filter convolutions on the image. Therefore, the most popular type of network to process images is the Convolutional Neural Network (CNN) where the linear functions are convolutions. In our case, we do not have any prior about the structure of the feature vectors and just consider them as vectors in $\mathbb{R}^n$. Therefore, our networks have a simple architecture where linear layers are simply layers called *fully connected* which are defined by $g_i(y) = W_i y + b_i$ where $y \in \mathbb{R}^{n_i}$ is the layer's input and $W_i \in \mathbb{R}^{n_{i+1}\times n_i}$ and $b_i \in \mathbb{R}^{n_{i+1}}$ are the parameters to be learned.

Numerous activation functions for neural networks have been proposed. Historically, the most popular functions were the sigmoid $g(y) = (1+e^y)^{-1}$ and the hyperbolic tangent. However, those functions saturate on both ends which makes neural networks hard to train because the gradient vanishes for large absolute values of y. Currently, the most used activation function is the Rectified Linear Unit (ReLU) which has been propose by He et al. in [56]. It simply consists in thresholding the negative values to 0: $ReLU(y) = \max(0, y)$. ReLU has two interesting properties: it does not saturate on both end and is very fast to compute. The use of ReLU produces sparse outputs which can also be an interesting property. Several variants of the ReLU have been proposed, such as the PReLU function [59]: $PReLU(y) = \max(0, y) - a\max(0, -y)$ where a is a trainable parameter. However, the original ReLU remains currently the most used activation function.

Another popular activation function is called maxout [60]. This function consists in computing the element-wise maximum between m inputs which come from the previous linear layer. Consequently, the output size of the maxout function is m times smaller than its input. For example, with $m = 2$, the maxout function can be written $g(y) = \max\left(y_{1\ldots n_i/2}, y_{1+n_i/2\ldots n_i}\right)$ where max is the element-wise maximum between two vectors. In Convolutional Neural Networks, one of the basic components are pooling layers which progressively reduce the spatial resolution of the feature maps. Maxout can be seen as type of pooling which acts across feature and not at the spatial level. The succession of one set of linear layers and a maxout is a universal approximator of any convex function [60]. A neural network with maxout where $m = 5$ recently obtained the state-of-the-art result on MNIST without artificial data augmentation [61]. In our experiments, we have evaluated two activation functions: ReLU and maxout.

The number of trainable parameters in a neural network can be very large. Therefore, one major challenge is to obtain good generalization performance. The amount of data used for training is obviously key and indeed, as we mentioned in the introduction of this chapter, neural networks started to perform really well when large labeled datasets became available. In addition, several regularization schemes have been proposed to limit over-fitting. For example, penalizing the L1-norm or L2-norm of the parameters is widely used (the L2-norm regularizer is often called *weight decay* in the neural network community). Recently, a new method called *dropout* has been introduced [57]. It consists in switching off a random subset of neurons while performing each gradient step to prevent complex co-adaptation of the network's parameters. The proportion of units switched off can be fairly large, a ratio of 50% is not unusual. The dropout is only active during training, no neurons are switched off when the network is used at test time. While using dropout significantly slows down the convergence of the network during training, it has often a great positive impact on generalization performance. It has also been observed that dropout leads to an increased sparsity of the activations. We tried several rates of dropout for each experiment and empirically selected the best one for the results presented in Section 5.3.

5.1.2 Intermediate Losses

During training, the evolution of the network's parameters are driven by the gradient of the objective function. The network's parameters are initialized randomly. At the beginning of the training, the gradients with respect to the parameters of the first layers of the network are obtained by backpropagation through the random weights of the last layers. Therefore, those gradients might carry little relevant information and it might make the training very slow or even sometimes making it fail. One method to avoid this problem is to add intermediate losses. This concept has been first introduced in [62] to train a 22-layers network which classifies images for the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). Each intermediate loss is associated with an auxiliary classifier which adds additional trainable weights.

In the context of similarity function learning, we propose a simpler scheme which does not add any additional parameter. We want our neural network to transform progressively the input features into more discriminative ones. After each linear layer, we directly plug a loss function similar to (5.1) and the overall objective function becomes a weighted sum of all the intermediate losses and the final one. The weights associated with the intermediate losses should be small (typically 0.05 or 0.1) because their role is just to push the network in a good direction at the beginning of the training but the end goal is to obtain discriminative features after the last layer. One might even consider removing the intermediate losses toward the end of the training.

5.2 Training Pair Selection for Deep Metric

In some applications, the training pairs are directly given. But most of the times, the training dataset provides a label per sample and the pairs need to be built. The number of possible dissimilar pairs grows quadratically with the dataset size and becomes quickly too large to be dealt with. In Section 3.1.5, we have described a pair selection mechanism that works well for learning a linear metric or a local metric such as LMLML. The pair selection is performed just once, before the actual training begins. One advantage of this one step method is that it preserves the convexity of the optimization problem used for learning linear and local metric.

However, as neural networks are much more powerful than linear models, they can take advantage of training with more pairs. We propose to draw new pairs regularly to adapt the pairs to the current state of the neural network. Neural network training being already non-convex, we do not lose any interesting property by applying several rounds of pair selection. Following this approach, the network is trained with increasingly hard pairs. This type of scheme has shown to be effective to learn complex models [63]. We have implemented two different pair selection approaches. Both methods select the similar pairs with a basic random sampling but differ by the procedure used to select dissimilar pairs.

The first method seeks dissimilar pairs which have approximately the same distance distribution as the similar pairs. We cannot rely on standard sampling algorithms, such as Metropolis-Hastings, as the procedure would be very slow because the distributions of similarity measures for similar and dissimilar pairs are very different. As an alternative, we propose the following procedure. We first randomly create a large pool of potential dissimilar pairs and compute their associated similarity. Then, we sort both pairs' lists with respect to their associated distances and then use Dynamic Time Warping (DTW) to select dissimilar pairs which have similarities close to those of the similar pairs. DTW is an algorithm designed to compute a distance between two time series. It aligns sequences by matching or deleting measurements to minimize a global alignment cost. We keep the dissimilar pairs which are matched with a similar pair by the DTW. For efficiency, we use a fast approximation of the DTW that has linear time complexity [64].

The second method is inspired by the triplet selection method presented in [24]. For every similar pair (x_i, x_j), we build a dissimilar pair (x_i, x_i') where x_i' is selected among a pool of candidates C by solving the problem $\min_{x_i' \in C} \left(\|f_\theta(x_i) - f_\theta(x_j)\| - \|f_\theta(x_i) - f_\theta(x_i')\| \right)^2$. The size of the set C is typically between 16 and 128 depending of how tough we want the dissimilar pairs to be. As a result, for each similar pair, we get a dissimilar one with one point in common and a distance close to that of the similar pair. In most of our experiments, this second method delivers better performance than that using DTW.

5.2.1 Separate Scale Optimization

The objective function (5.1) is not scale invariant, its goal is making 1 a good threshold to separate similar and dissimilar pairs. At the beginning of the training the network's parameters are first modified so that the average pair distance is 1. We observed that this adjustment is mainly obtained by changing the weights of the last linear layer. The consequence is that the L2-norm of the last layer becomes very different from that of the other layers. This can cause a problem when L2-norm regularization is applied because the regularization's intensity is going to be different for the last layer and the others.

To solve these two issues, we propose to add an additional scalar parameter $\alpha > 0$ which rescales the squared distance $\|f_\theta(x_i) - f_\theta(x_j)\|^2$ in the objective function. The objective function (5.1) becomes

$$\min_{\alpha,\theta} \sum_{(i,j) \in \mathcal{T}} \ell_\gamma \left(r_{ij}, \alpha \|f_\theta(x_i) - f_\theta(x_j)\|^2 \right). \tag{5.2}$$

The training is done in two steps. First, the objective function is minimized with respect to α only. Second, the rest of the network is trained while α is kept fixed. When intermediate losses are used (see previous section), one scale parameter should be added for each loss.

The full objective function to minimize, which combines regularization, intermediate losses and separate scale optimization is given by:

$$\min_{\theta,\{\alpha_1,\dots,\alpha_K\}} \sum_{(i,j)\in\mathcal{T}} \left(\ell_\gamma\left(r_{ij}, \alpha_K \|f_\theta(x_i) - f_\theta(x_j)\|^2\right) + \beta \sum_{k\in IL} \ell_\gamma(r_{ij}, \alpha_k \|f_\theta^k(x_i) - f_\theta^k(x_j)\|^2) \right) + \lambda \sum_{l\in L} \|W_l\|_F^2 \tag{5.3}$$

where $f_\theta^k = g_1 \circ g_2 \circ \cdots \circ g_k$ corresponds to the composition of the k first layers of the network, IL is the set of all intermediate linear layers, L is the set of all linear layers, β is the weight applied to intermediate losses and λ tunes the regularization strength. We first minimize it with respect to $\{\alpha_k\}_{k=1\dots K}$ and then to θ.

There is a second advantage in optimizing separately the variables $\{\alpha_k\}_{k=1\dots K}$ and the weights of the network. As explained in the previous subsection, there is an interest in performing several rounds of pair selection during training. If the scales are not adjusted after replacing the training set by a new one with harder dissimilar pairs, the training is perturbed because the average distance of dissimilar pairs makes a sudden jump to a smaller value. This creates instability in the learning process and slows down the training.

After each pair selection step, we re-optimize the scales $\{\alpha_k\}_{k=1\dots K}$ before continuing the optimization of the network. Using this separate scale optimization scheme leads to a significant performance improvement on several datasets. For example, we obtain a 9% relative increase of the classification rate on Isolet (see Section 5.3 for experimental results).

5.3 Experiments

We have used the TensorFlow framework to conduct our experiments with Deep Metric. The design of a neural network which works well for a specific task is rather heuristic and usually requires several trials and errors. We have evaluated different neural network architectures for each dataset and present the performance of the best configurations in Table 5.2 and Figure 5.1. Table 5.1 summarizes the network configuration used for each dataset. The dimensionality of the output feature vectors are equal to the dimensionality of the input to make the results of this section comparable to those presented in Chapter 4. The column "Hidden layers size" indicates the size of each

Table 5.1: Network configurations. In the Activation function column, MO stands for maxout.

	Input / output size	Hidden layers size	Activation function	Dropout rate	λ	β
MNIST	164	[500]	MO $m = 2$	20%	0.1	0.05
Isolet	171	[256, 256]	ReLU	10%	0.01	0.05
Letter	16	[128, 256]	MO $m = 2$	0%	0	0.05
20newsgroup	1000	[1000, 1000]	ReLU	10%	0.01	0.05
Reuters	100	[250]	ReLU	20%	0.1	0.05
FRGC	700	[700]	ReLU	0%	0.01	0.05

Table 5.2: Classification Rates

	MNIST	Isolet	Letter	20newsgroup	Reuters
Deep Metric	**98.61%**	**96.79%**	97.49%	**78.61%**	**89.07%**
LMLML	98.10%	96.02%	**97.60%**	77.06%	89.03%
LMLML $K = 0$	97.86%	94.74%	97.21%	76.28%	88.75%

hidden layer (i.e. the list length is equal to the number of hidden layers).

Deep Metric is better that the linear metric (LMLML with $K = 0$) on all datasets and outperforms the local metric method LMLML on five datasets out of six. On

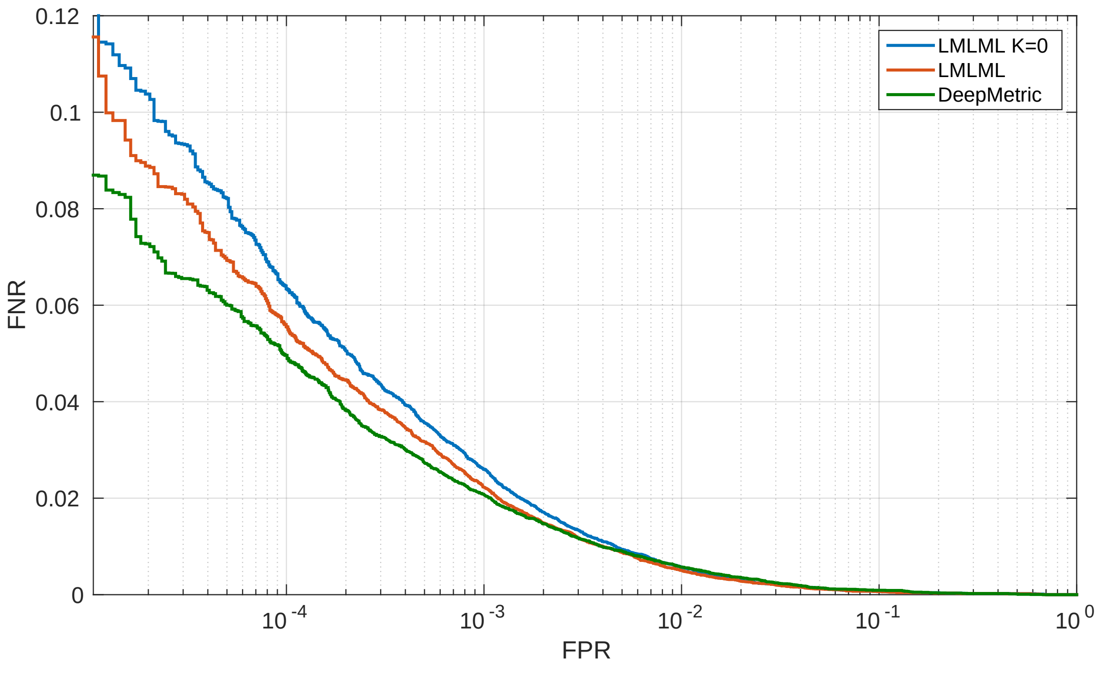


Figure 5.1: DET curve of LMLML and Deep Metric on FRGC. The red curve corresponds to the performance with $K = 3$ which is the value giving the best results on FRGC.

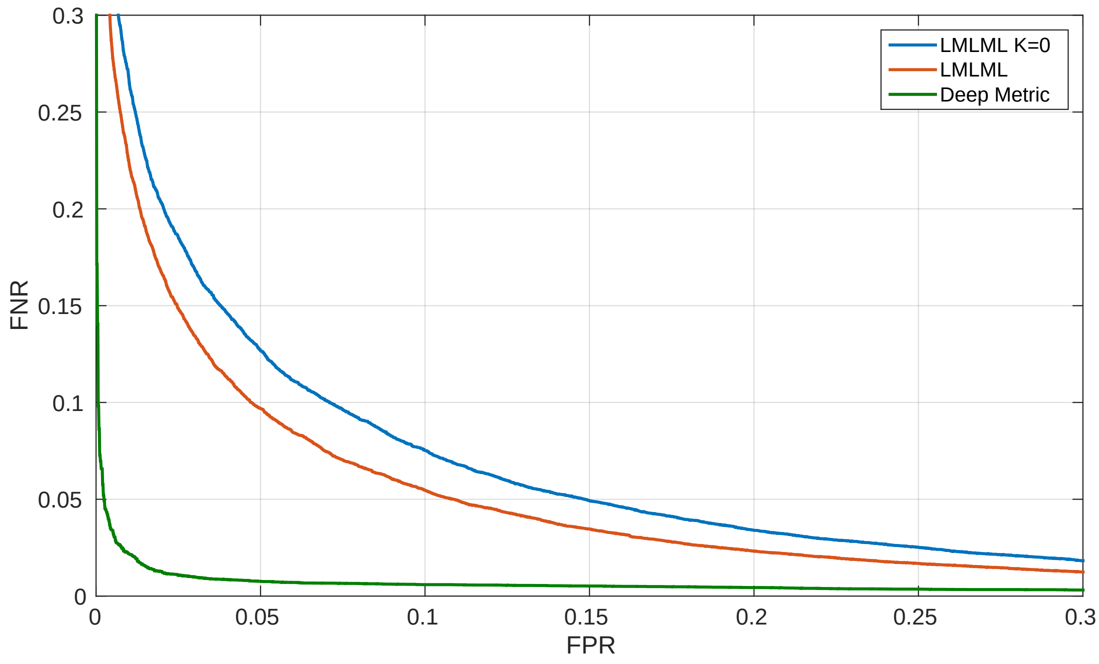


Figure 5.2: DET curve on MNIST for LMLML and Deep Metric.

the classification datasets, the performance reported in Table 5.2 is the classification accuracy obtained when the similarity functions are combined with a nearest neighbor classifier. This classifier is very simple but powerful. It can create highly non-linear class boundaries, especially when the number feature vectors in the reference set is large. To lead to good performance, a similarity function used in a nearest neighbor classifier is only required to work well at very short range.

Linear metrics are linear similarity functions. LMLML is a non-linear similarity function which adjusts the similarity to local specificities. Both are unable to perform well with similar pairs composed of feature vectors coming from very different parts of the space. For example, they could not solve the XOR problem. On the other hand, Deep Metric is perfectly able to deal with this problem. To compare Deep Metric and LMLML on MNIST without using a nearest neighbor classifier, we computed the DET curves (Figure 5.2). Deep Metric obtains an EER of 1.5% whereas LMLML gets 7.2% and a linear metric 8.7%. The gap is much larger than when we look at the classification accuracy because Deep Metric can output small distances for pairs of digits of the same class with very different appearances (examples are given in Figure 5.3)

We also tried using Deep Metric on LFW but did not obtain competitive results. We think that there are two main reasons for this failure. First, LFW is composed of

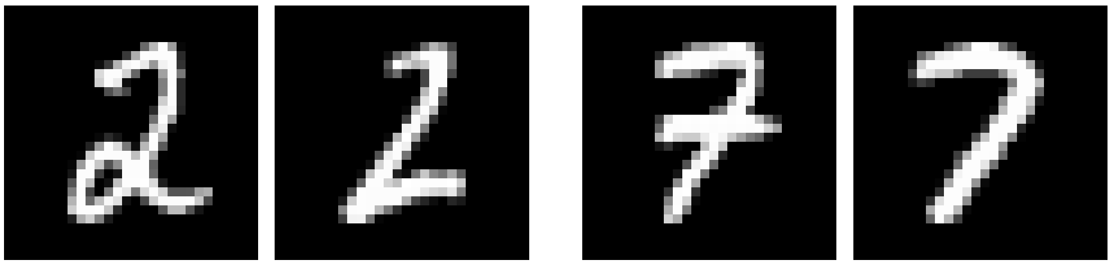


Figure 5.3: These two pairs of digits are considered to be very similar by Deep Metric but not by LMLML. Deep Metric handles similar pairs of feature vectors which are too far away in the original feature space better than local metrics.

only 13233 images which is too few for learning a task as complex as face verification with deep learning. Second, we used a feature extractor based on LBP histograms which might not produce rich enough features to benefit from deep methods.

5.4 Conclusion

One strength of modern Deep Learning approaches like CNN is that they create a hierarchical feature transform by taking advantage of the structure of the data. When applied on data without any specific structure such as feature vectors, we cannot obtain such kind of hierarchical representations. Nonetheless, we have shown that fully connected neural networks with one or two hidden layers obtain state-of-the-art results in similarity function learning when the amount of available training data is large enough.

Chapter 6

Similarity Function Learning with Data Uncertainty

Standard similarity functions compute a similarity value from a pair of feature vectors. Feature vectors are often corrupted by noise and this contributes to make similarity learning a challenging task. In this chapter, we show how knowledge about the uncertainty of each feature of each data point could be used when available. For example, the uncertainty of a local image descriptor in the top left corner of an image could depend on the signal to noise ratio in that area. It would be different from one image to another and independent of the signal to noise ratio in, say, the bottom right corner. Nonetheless, this uncertainty information is ignored by most machine learning algorithms which simply treat each sample as a *point* in a feature space. To overcome this limitation, we propose to consider each sample as a probability distribution whose parameters are provided by the feature extraction process. Each sample has a specific distribution which reflects the uncertainty in the corresponding features. This idea has been explored for some machine learning tasks. Particularly, several classification algorithms have been extended to deal with uncertain data, including support vector machines [65, 66], decision trees [67], and naive Bayes classifier [68]. Clustering algorithms have also been adapted to uncertain data, see, for example, [69, 70] and references therein.

Up to our knowledge, this type of approach has never been applied to similarity function learning. We extend the Joint Bayesian method [17] presented in Section 1.3 to deal with uncertainty information. Our approach is composed of an unsupervised dimensionality reduction stage and the similarity function itself. Uncertainty is taken into account throughout the whole processing pipeline during both training and testing.

We have published the method described in this chapter in a ICPRAM16 paper [71].

This chapter is organized as follows. We start by proposing an uncertainty-aware dimensionality reduction algorithm inspired by the PPCA [72] in Section 6.1. Then, we detail how to compute our uncertainty-aware similarity function and how to learn its parameters in Section 6.2. Section 6.3 presents experiments which indicate that our uncertainty-aware similarity function outperforms standard similarity function learning methods on challenging tasks. Finally, we summarize our findings in Section 6.4.

6.1 Dimensionality Reduction

In many fields such as computer vision, feature vectors are often very high dimensional. Therefore, most similarity function methods start with a dimensionality reduction step in order to limit the computational cost and limit the risk of over-fitting. PCA has been shown to be both simple and effective for this task but does not take into account any uncertainty information. In this section we propose a dimensionality reduction method which uses the uncertainty information to learn the low dimensional space and to project new feature vectors into it.

6.1.1 Uncertainty-Aware Probabilistic PCA

Our dimensionality reduction method, Uncertainty-Aware Probabilistic PCA (UA-PPCA), uses a generative model similar to that used in Probabilistic PCA [72] or Factor Analysis. This latent variable model defines the observation $\widetilde{x}$ as the sum of a linear transformation of a low dimensional latent variable x plus some noise. x is assumed to follow the standard multivariate normal distribution $\mathcal{N}(0, I)$. Specifically, our model can be written as

$$\widetilde{x} = \mu + Wx + \widetilde{\varepsilon}_x \tag{6.1}$$

where $\widetilde{x} \in \mathbb{R}^n$, $\mu \in \mathbb{R}^n$ is the center of the observation space, $W \in \mathbb{R}^{n \times m}$ relates the observation and the latent space, $x \in \mathbb{R}^m$ and $\widetilde{\varepsilon}_x \in \mathbb{R}^n$ is a Gaussian noise of distribution $\mathcal{N}(0, \widetilde{S}_x)$. The uncertainty associated with the feature vector $\widetilde{x}$ is represented by the covariance matrix $\widetilde{S}_x$.

The difference between PPCA or Factor Analysis and our method is that we make a different assumption on the noise distribution. In PPCA and Factor Analysis, a single noise covariance matrix is common to all samples. This makes it possible to learn this matrix from the data. In contrast, in UA-PPCA, each vector $\widetilde{\varepsilon}_x$ has its own covariance matrix $\widetilde{S}_x$ which reflects the uncertainty in each component of the specific feature vector $\widetilde{x}$. The matrices $\widetilde{S}_x$ all being different, they cannot be learned and therefore have to be provided by the feature extractor. They are regarded as fixed during the learning process.

Considering that two features are uncorrelated is very different from saying that the noise which affects them are uncorrelated. In a picture of a face, the appearance of the two eyes are obviously correlated. However, the noise affecting them in a given image can very well be different if, for example, there is a cast shadow on one side of the face. In the rest of this chapter, we assume that the noise is uncorrelated and therefore consider that the covariance matrices $\widetilde{S}_x$ are diagonal.

Usually, dimensionality reduction consists in finding low dimensional projections corresponding to high dimensional data. In the context of uncertainty-aware similarity function, the whole probability distribution of $\widetilde{x}$ needs to be transferred into the low dimensional space. Following our generative model, the low dimensional projection x and its associated uncertainty are respectively the mean and the covariance matrix of the conditional probability distribution $P(x|\widetilde{x},\widetilde{S}_x,W,\mu)$. Using Bayes theorem and the Gaussian product rule, we obtain the closed-form formula:

$$P(x|\widetilde{x},\widetilde{S}_x,\mu,W) = \mathcal{N}(x|\mu_x,S_x) \tag{6.2}$$

where
$$S_x = (W^\top \widetilde{S}_x^{-1} W + I)^{-1} \tag{6.3}$$

and
$$\mu_x = S_x W^\top \widetilde{S}_x^{-1} (\widetilde{x} - \mu). \tag{6.4}$$

6.1.2 Learning μ and W

In this subsection we present an Expectation-Maximization algorithm (EM) to learn the parameters of the model $\Theta = \{\mu, W\}$ from an unlabeled training dataset composed of feature vectors $\widetilde{x}_i \in \mathbb{R}^n$ and their associated diagonal covariance matrices $\widetilde{S}_i \in \mathbb{R}^{n\times n}$.

The EM algorithm is composed of two steps performed alternatively. The Expectation step (E-step) consists in estimating the parameters of the distribution of the latent variables x_i given the previous estimate of the parameters $\bar{\Theta}$. During the Maximization step (M-step), we maximize $Q(\Theta,\bar{\Theta})$, the expectation over the latent variables of the log-likelihood of the complete data, with respect to Θ. Its expression is given by

$$Q(\Theta,\bar{\Theta}) = \sum_i \int P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta}) \left(\log P(x_i) + \log P(\widetilde{x}_i|x_i,\widetilde{S}_i,\Theta)\right) \mathrm{d}x_i \tag{6.5}$$

$$= -\frac{1}{2}\sum_i \int P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta})\,(\widetilde{x}_i - \mu - Wx_i)^\top \widetilde{S}_i^{-1}\,(\widetilde{x}_i - \mu - Wx_i)\,\mathrm{d}x_i + const \tag{6.6}$$

where *const* is a term which does not depend on Θ and can therefore be ignored.

During the E-step, we estimate the parameters of the distributions of the latent variables $P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta})$ using equation (6.2). The M-step, namely the maximization of Q with respect to Θ, is achieved by solving the system of equations $\partial Q(\Theta,\bar{\Theta})/\partial\Theta = 0$. Specifically,

$$\frac{\partial Q(\Theta,\bar{\Theta})}{\partial \mu} = \sum_i \widetilde{S}_i^{-1}\left(\widetilde{x}_i - \mu - W\int P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta})x_i\,\mathrm{d}x_i\right)$$

$$= \sum_i \widetilde{S}_i^{-1}\,(\widetilde{x}_i - \mu - W\mu_{x_i}) \tag{6.7}$$

and

$$\frac{\partial Q(\Theta,\bar{\Theta})}{\partial W} = \sum_i \widetilde{S}_i^{-1}\left((\widetilde{x}_i - \mu)\int P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta})x_i^\top\,\mathrm{d}x_i - W\int P(x_i|\widetilde{x}_i,\widetilde{S}_i,\bar{\Theta})x_ix_i^\top\,\mathrm{d}x_i\right)$$

$$= \sum_i \widetilde{S}_i^{-1}\left((\widetilde{x}_i - \mu)\mu_{x_i}^\top - W\left(S_{x_i} + \mu_{x_i}\mu_{x_i}^\top\right)\right). \tag{6.8}$$

There is no closed-form solution for this system of equations in the general case. However, in our model we restrict the uncertainty covariance matrices $\widetilde{S}_i$ to be diagonal. In this case, we obtain a closed-form solution for each component of μ and each row of W which is given by

$$\mu^{(j)} = \frac{\left(\sum_i \frac{\widetilde{x}_i^{(j)}}{\widetilde{S}_i^{(j,j)}}\mu_{x_i}\right)^\top A_j a_j - \sum_i \frac{\widetilde{x}_i^{(j)}}{\widetilde{S}_i^{(j,j)}}}{a_j^\top A_j a_j - \sum_i \frac{1}{\widetilde{S}_i^{(j,j)}}} \tag{6.9}$$

$$W^{(j,\cdot)} = \left(\sum_i \frac{\widetilde{x}_i^{(j)}}{\widetilde{S}_i^{(j,j)}}\mu_{x_i} - \mu^{(j)}a_j\right)^\top A_j \tag{6.10}$$

where $$A_j = \left(\sum_i \frac{S_{x_i} + \mu_{x_i}\mu_{x_i}^\top}{\widetilde{S}_i^{(j,j)}}\right)^{-1}, \tag{6.11}$$

$$a_j = \sum_i \frac{1}{\widetilde{S}_i^{(j,j)}} \mu_{x_i}, \tag{6.12}$$

$(\cdot)^{(j,j)}$ denotes the jth element of the diagonal of a matrix, $(\cdot)^{(j,\cdot)}$ its jth row and $(\cdot)^{(j)}$ the jth component of a vector. The parameters μ and W have to be initialized before the first iteration of the EM algorithm. We simply initialize μ to the empirical mean of the data and W to the m first leading eigenvectors of the empirical covariance matrix of the training set multiplied by the square-root of their respective eigenvalue. The computational complexity of each EM iteration is $\mathcal{O}(n(m^3 + Nm^2)$ where n and m are respectively the dimensionality of the original and low dimensional feature vectors and N is the number of training samples.

6.2 Uncertainty-Aware Similarity Function

In this section, we present our similarity function: Uncertainty-Aware Likelihood Ratio (UA-LR). The feature vectors and their associated uncertainty covariance matrices used in this section are usually the outputs of the dimensionality reduction method presented in the previous section. However, when the dimensionality of the original feature space is not too large, we can bypass the dimensionality reduction stage and directly apply the similarity function. We start by describing the generative model. The associated similarity function is presented in Section 6.2.2. Finally, in Section 6.2.3, we propose an EM-based algorithm to learn the model parameters.

6.2.1 Generative Model

Gaussian generative models are popular because they are both relatively simple and effective. Many face recognition algorithms rely on Gaussian assumptions such as FisherFaces [4], KISSME [5], Joint Bayesian Faces [17], and PLDA [73]. Those approaches model the data as the sum of two terms, $x = \mu_c + \delta$, where μ_c is the center of the class to which x belongs to and δ is the deviation relative to its class center. We propose to split δ into two further terms, leading to the following model:

$$x = \mu_c + w + \varepsilon_x \tag{6.13}$$

where w is the intrinsic variation of the sample from its class center μ_c and ε_x is an observation noise. As opposed to the previous methods, this model explicitly takes the uncertainty information into account by considering that it affects the distribution of ε_x. All those variables follow zero-mean multivariate normal distributions: $\mu_c \sim \mathcal{N}(0, S_\mu)$, $w \sim \mathcal{N}(0, S_w)$ and $\varepsilon_x \sim \mathcal{N}(0, S_x)$. In the remaining of this chapter, S_μ is called between-class covariance matrix, S_w within-class covariance matrix and S_x uncertainty covariance matrix.

S_μ and S_w are common to all samples and are unknown. We introduce an EM algorithm to estimate them in Section 6.2.3. On the contrary, S_x is specific to each feature vector and is either computed by the Uncertainty-Aware Probabilistic PCA described in the previous section from the original feature vectors $\widetilde{x}$ and their uncertainty covariance matrices $\widetilde{S_x}$ or, directly provided by the feature extractor when dimensionality reduction is not needed. The uncertainty matrix of the original input features $\widetilde{S_x}$ is always diagonal but, after dimensionality reduction, the matrix S_x computed with (6.4) is a full covariance matrix.

6.2.2 Uncertainty-Aware Likelihood Ratio

In Bayesian decision theory, decisions based on thresholding the likelihood ratio are known to achieve minimum error rate (Neyman-Pearson lemma). In this method, we use the log-likelihood ratio associated with the above generative model as our similarity function.

Two feature vectors belonging to the same class (similar pair hypothesis: $\mathrm{H_{sim}}$) share the same value for μ_c and only differ in their respective intrinsic variation w and observation noise ε_x. In contrast, two vectors from different classes (dissimilar pair hypothesis: $\mathrm{H_{dis}}$) are totally independent.

Let x_i and x_j be two feature vectors and S_i and S_j their associated uncertainty covariance matrices. Following the same methodology as in [17] we derive the probability distributions $P(x_i, x_j | \mathrm{H_{sim}}, S_i, S_j)$ and $P(x_i, x_j | \mathrm{H_{dis}}, S_i, S_j)$ from the generative model:

$$P(x_i, x_j | \mathrm{H_{sim}}, S_i, S_j) = \mathcal{N}(\left[x_i^\top x_j^\top\right]^\top | 0, S_{\mathrm{sim}}) \tag{6.14}$$

$$P(x_i, x_j | \mathrm{H_{dis}}, S_i, S_j) = \mathcal{N}(\left[x_i^\top x_j^\top\right]^\top | 0, S_{\mathrm{dis}}) \tag{6.15}$$

where

$$S_{\text{sim}} = \begin{bmatrix} S_\mu + S_w + S_i & S_\mu \\ S_\mu & S_\mu + S_w + S_j \end{bmatrix} \tag{6.16}$$

and

$$S_{\text{dis}} = \begin{bmatrix} S_\mu + S_w + S_i & 0 \\ 0 & S_\mu + S_w + S_j \end{bmatrix}. \tag{6.17}$$

The log-likelihood ratio $LR(x_i,x_j|S_i,S_j) = \log\left(\frac{P(x_i,x_j|\mathrm{H}_{\text{sim}},S_i,S_j)}{P(x_i,x_j|\mathrm{H}_{\text{dis}},S_i,S_j)}\right)$ is obtained in closed-form using block-wise inversion matrix and determinant formula. Specifically, a direct computation gives

$$\begin{aligned} LR\left(x_i,x_j|S_i,S_j\right) = x_i^\top \left(M_1 - \left(S_\mu + S_w + S_i\right)^{-1}\right) x_i + x_j^\top \left(M_3 - \left(S_\mu + S_w + S_j\right)^{-1}\right) x_j \\ + 2x_i^\top M_2 x_j - \log\left|S_\mu + S_w + S_i\right| - \log\left|M_1\right| + const \end{aligned} \tag{6.18}$$

where

$$M_1 = \left(S_\mu + S_w + S_i - S_\mu \left(S_\mu + S_w + S_j\right)^{-1} S_\mu\right)^{-1}, \tag{6.19}$$

$$M_2 = -M_1 S_\mu \left(S_\mu + S_w + S_j\right)^{-1}, \tag{6.20}$$

$$M_3 = \left(S_\mu + S_w + S_j\right)^{-1} \left(I - S_\mu M_2\right) \tag{6.21}$$

and *const* is a constant which does not depend on either x_i, x_j, S_i or S_j and can therefore be ignored.

The similarity function is a quadratic form of the feature vectors x_i and x_j. The contribution of a specific component of the feature vectors to the similarity score depends on two factors: its discriminative power which is function of S_μ and S_w, and its reliability which is measured by S_i and S_j. The Uncertainty-Aware Likelihood Ratio presented in this section combines these different types of information to compute a meaningful similarity.

6.2.3 Parameters Estimation

The parameters of our model to learn are the covariance matrices S_μ and S_w. We present an EM algorithm to estimate them in this subsection.

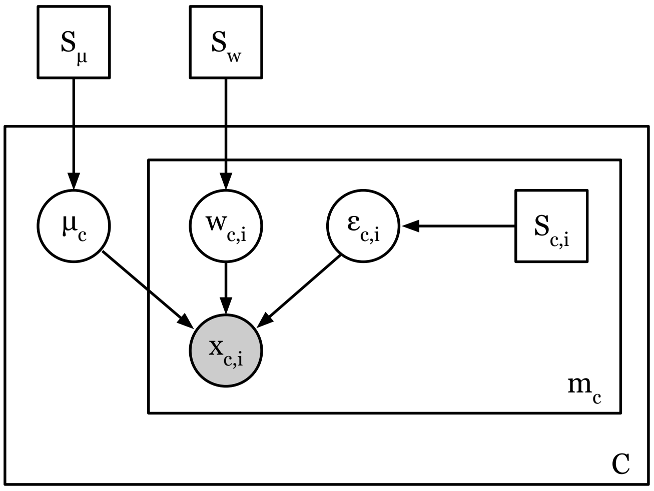


Figure 6.1: Graphical representation of the generative model using plate notation. All the covariance matrices S_μ, S_w and $S_{c,i}$ are considered fixed in the generative model. However, while the matrices $S_{c,i}$ are provided by UA-PPCA or the feature extractor, the matrices S_μ and S_w are estimated by the EM algorithm.

We consider a training set with C different classes. Any class c contains m_c feature vectors, $x_{c,1},\ldots,x_{c,m_c}$. X_c denotes the concatenation of those feature vectors and $S_{x_{c,1}},\ldots,S_{x_{c,m_c}}$ their respective uncertainty covariance matrices. We define the latent variables $Z_c = \{\mu_c, w_{c,1},\ldots,w_{c,m_c}\}$ and the parameters to estimate $\Psi = \{S_\mu, S_w\}$. The graphical representation of the generative model of the dataset is depicted in Figure 6.1. The EM algorithm consists in iteratively maximizing $R(\Psi,\bar{\Psi})$, the expectation of the log-likelihood of the complete data over the latent variables Z_c given the previous estimate of the parameter $\bar{\Psi}$. $R(\Psi,\bar{\Psi})$ is given by

$$R(\Psi,\bar{\Psi}) = \sum_{c=1}^{C} \int P(Z_c|X_c,\bar{\Psi}) \log P(X_c,Z_c|\Psi)\,\mathrm{d}Z_c \tag{6.22}$$

$$= \sum_{c=1}^{C} \int P(Z_c|X_c,\bar{\Psi}) \Big(\log P(\mu_c|S_\mu) + \sum_{i=1}^{m_c} \log P(w_{c,i}|S_w) + \sum_{i=1}^{m_c} \log P(x_{c,i}|\mu_c, w_{c,i}, S_{c,i}) \Big)\,\mathrm{d}Z_c. \tag{6.23}$$

The standard E-step would consist in estimating the parameters of the distribution $P(Z_c|X_c,\bar{\Psi})$. But Z_c might have a very high dimensionality especially for classes containing a large number of samples and therefore manipulating directly the parameters of $P(Z_c|X_c,\bar{\Psi})$ could be a heavy computational burden. In order to make the optimization computationally tractable, we take advantage of the structure of the problem.

Namely, we observe that the latent variables $w_{c,i}$ are conditionally independent among themselves given μ_c (see Figure 6.1). Therefore $P(Z_c|X_c,\bar{\Psi})$ can be factorized as:

$$P(Z_c|X_c,\bar{\Psi}) = P(\mu_c|X_c,\bar{\Psi})\prod_{i=1}^{m_c} P(w_{c,i}|x_{c,i},\mu_c,\bar{\Psi}). \tag{6.24}$$

To maximize $R(\Psi,\bar{\Psi})$ with respect to Ψ, we solve the equation $\partial R(\Psi,\bar{\Psi})/\partial\Psi = 0$. The optimal values for S_μ and S_w can be computed separately and we explicit the update formula in the next two sections.

6.2.3.1 Update of S_μ

As shown in the next paragraph, the solution for S_μ depends on the parameters of the distribution $P(\mu_c|X_c,\bar{\Psi})$ which is a normal distribution $\mathcal{N}(\mu_c|b_{\mu_c},T_{\mu_c})$ where

$$T_{\mu_c} = \left(\bar{S_\mu}^{-1} + \sum_{i=1}^{m_c}\left(\bar{S_w}+S_{c,i}\right)^{-1}\right)^{-1} \quad \text{and} \tag{6.25}$$

$$b_{\mu_c} = T_{\mu_c}\sum_{i=1}^{m_c}\left(\bar{S_w}+S_{c,i}\right)^{-1} x_{c,i}. \tag{6.26}$$

The proof is given in the supplementary material Section A.1. It is interesting to observe how uncertainty impacts the probability distribution of μ_c. For samples with very large uncertainty, $\left(\bar{S_w}+S_{c,i}\right)^{-1}$ becomes close to the null matrix and therefore these samples have little weight in the computation of T_{μ_c} and b_{μ_c}. This weighting operates at the feature level, meaning that a given sample can have a small weight for some features and a large one for others.

To find the matrix S_μ maximizing $R(\Psi,\bar{\Psi})$, we compute its gradient with respect to S_μ:

$$\begin{aligned}
\frac{R(\Psi,\bar{\Psi})}{\partial S_\mu} &= \sum_{c=1}^{C}\int P(Z_c|X_c,\bar{\Psi})\frac{\partial \log P(\mu_c|S_\mu)}{\partial S_\mu}\,\mathrm{d}Z_c \\
&\propto \sum_{c=1}^{C}\int P(\mu_c|X_c,\bar{\Psi})\left(S_\mu^{-1} - S_\mu^{-1}\mu_c\mu_c^\top S_\mu^{-1}\right)\mathrm{d}\mu_c && (6.27)\\
&\propto CS_\mu^{-1} - S_\mu^{-1}\sum_{c=1}^{C}\int P(\mu_c|X_c,\bar{\Psi})\mu_c\mu_c^\top\,\mathrm{d}\mu_c S_\mu^{-1} && (6.28)\\
&\propto CS_\mu^{-1} - S_\mu^{-1}\sum_{c=1}^{C}\left(T_{\mu_c} + b_{\mu_c}b_{\mu_c}^\top\right)S_\mu^{-1} && (6.29)
\end{aligned}$$

from which we obtain the following closed-form update formula

$$S_\mu = \frac{1}{C}\sum_{c=1}^{C}\left(T_{\mu_c} + b_{\mu_c} b_{\mu_c}^\top\right). \tag{6.30}$$

6.2.3.2 Update of S_w

The optimization of $R(\Psi,\bar{\Psi})$ with respect to S_w requires the knowledge of the parameters of the distribution $P(w_{c,i}|X_c,\bar{\Psi})$. We show in the supplementary material Section A.2 that $P(w_{c,i}|X_c,\bar{\Psi}) = \mathcal{N}(w_{c,i}|b_{w_{c,i}},T_{w_{c,i}})$ where

$$T_{w_{c,i}} = R_{c,i}S_{c,i}^{-1}T_{\mu_c}S_{c,i}^{-1}R_{c,i} + R_{c,i}, \tag{6.31}$$

$$b_{w_{c,i}} = R_{c,i}S_{c,i}^{-1}\left(x_{c,i} - b_{\mu_c}\right) \quad \text{and} \tag{6.32}$$

$$R_{c,i} = \left(S_{c,i}^{-1} + \bar{S_w}^{-1}\right)^{-1}. \tag{6.33}$$

The impact of $S_{c,i}$ on the parameters of the distribution is quite natural. If the uncertainty is large, the posterior probability $P(w_{c,i}|X_c,\bar{\Psi})$ converges to the prior $\mathcal{N}(w_{c,i}|0,\bar{S_w})$. This makes sense as, in the absence of a reliable observation, the prior should be used. However, if the uncertainty is very small then $P(w_{c,i}|X_c,\bar{\Psi})$ converges to $\mathcal{N}(w_{c,i}|x_{c,i} - b_{\mu_c}, T_{\mu_c})$ which does not depend anymore on the prior over $w_{c,i}$.

To maximize $R(\Psi,\bar{\Psi})$ with respect to S_w we compute its gradient which is given by

$$\frac{R(\Psi,\bar{\Psi})}{\partial S_w} = \sum_{c=1}^{C}\int P(Z_c|X_c,\bar{\Psi})\sum_{i=1}^{m_c}\frac{\partial \log P(w_{c,i}|S_w)}{\partial S_w}\,\mathrm{d}Z_c \tag{6.34}$$

and find the value of the matrix S_w which sets it to 0. The calculation uses the factorization (6.24) and is detailed in Section A.3 of the supplementary material. It leads to the following closed-form update equation

$$S_w = \frac{1}{\sum_{c=1}^{C} m_c}\sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}}\left(T_{w_{c,i}} + b_{w_{c,i}}b_{w_{c,i}}^\top\right). \tag{6.35}$$

6.2.3.3 Parameter Estimation Overview

EM algorithms need an initial estimate of the parameters to begin with. We initialize S_μ and S_w with their respective empirical estimate. To this end, we compute the empirical mean of each class, set S_μ to the covariance matrix of the means and S_w to the covariance matrix of the difference of each sample with the mean of its class. After initialization, we alternate between the E-step: the computation of the parameters T_{μ_c}, b_{μ_c}, $T_{w_{c,i}}$ and $b_{w_{c,i}}$ using equations (6.25), (6.26), (6.31) and (6.32) and the M-Step: the update of S_μ and S_w using equations (6.30) and (6.35). This process is repeated until the Frobenius norm of the differences between two consecutive estimates of S_μ and S_w are both smaller than a predefined threshold. The complexity of each iteration of the EM algorithm is $\mathcal{O}(Nm^3)$ where m is the feature vector dimensionality and N the number of training samples.

If we consider the data to be noiseless, our method is equivalent to the Joint Bayesian method [17]. In the case where the uncertainty of the features is unknown, considering the data to be slightly uncertain is a means to regularize the learning process. Indeed, by looking at the update equations of the matrices S_μ and S_w, we see that using constant uncertainty matrices $S_{c,i} = \alpha I$ with $\alpha > 0$ would act as a kind of Tikhonov regularization. The link between considering the data to be uncertain or noisy and regularization is well known [74]. This is the reason why performing artificial data augmentation by perturbing the original data is a common practice in machine learning, especially in the neural network community.

6.3 Experiments

The experiments presented in this section demonstrate the performance of the Uncertainty-Aware PPCA and the Uncertainty-Aware Likelihood Ratio. We first present results on MNIST to which we artificially add noise. Second, we show how the use of uncertainty can contribute to tack challenges in a real world application like face verification.

6.3.1 MNIST

Performance on MNIST is usually measured by classification accuracy and similarity functions are commonly combined with a nearest neighbor classifier to perform the actual classification. Our aim is to investigate the impact of noise and uncertainty on the performance of similarity functions. To evaluate solely similarity functions, we have conducted a digit verification experiment (given a pair of images, do they contain the same digit?) and report the Equal Error Rate (EER).

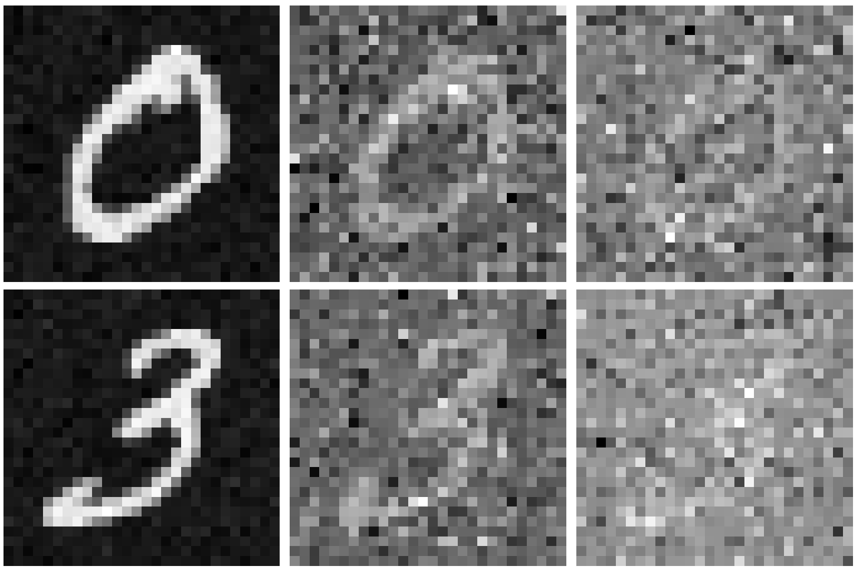


Figure 6.2: Examples of digits with three levels of additional noise: none (left), medium (middle) and strong (right)

We artificially add noise to the images to create uncertain data. The data generation protocol takes two steps: first, for each image and each pixel p, the noise standard deviation σ_p is drawn from a uniform law between 0 and t and second, we add to each pixel a noise drawn from a centered normal distribution with standard deviation σ_p. The uncertainty matrix associated to an image is simply the diagonal matrix containing the σ_p^2 of this image. By varying the value of t, we simulate different noise levels. Figure 6.2 shows examples of two different images affected by the three levels of noise we tested: none, medium and strong.

We compare our method, Uncertainty-Aware Likelihood Ratio (UA-LR) to three other methods: Joint Bayesian [17] (JB) to which our method is equivalent in the absence of noise, ITML [12] and the metric learned with the method based on the hinge loss and the Frobenius regularizer described in Section 3.2 (denoted HL+FR in the remaining of this chapter). We start by reducing the dimensionality to 100 using UA-PPCA for UA-LR and standard PCA for the three others as prescribed by the authors. As we can see in Table 6.1, the proposed method does not get the best results on noiseless data, however, thanks to the use of the uncertainty information, it outperforms the other methods on noisy data. Whereas the error rates of the other methods are more than doubled when a strong noise is added, UA-LR's EER relative increase is only of 46%.

	Methods			
Noise Level	UA-LR	JB	ITML	HL+FR
None	10.1%	10.1%	9.1%	**8.7%**
Medium	**12.2%**	13.5%	12.9%	12.5%
Strong	**14.7%**	20.6%	19.4%	18.8%

Table 6.1: EER on MNIST

	Perturbation intensity		
Noise Level	None	+/-30%	+/-60%
Medium	12.2%	12.3%	12.8%
Strong	14.7%	14.9%	16.3%

Table 6.2: Sensitivity to the uncertainty accuracy

In real applications, the exact values of the uncertainty values are unknown and only estimates can be provided to our algorithm. To evaluate its sensitivity to the accuracy of the uncertainty values, we propose to artificially perturb each σ_p by multiplying it by a factor uniformly drawn from $[0.7, 1.3]$ (for light perturbation) or $[0.4, 1.6]$ (for strong perturbation). Table 6.2 shows that our method is robust to this perturbation as the error rates increase of less than 11% even when a strong perturbation is applied.

In Section 6.1 we have proposed a new dimensionality reduction method named UA-PPCA which takes uncertainty into account. We evaluate the performance of UA-LR if we use the standard PCA instead of the proposed method to compute the matrix W and μ and/or if we replace the projection described in Section 6.1.1 using $P(x|\widetilde{x}, \widetilde{S_x}, W, \mu)$ by orthogonal linear projections ($W\widetilde{x}$ for feature vectors and $W^\top \widetilde{S_x} W$ for uncertainty matrices). Ignoring uncertainty at the dimensionality reduction stage leads to higher error rates (see Table 6.3). UA-LR does not even bring any improvement over the Joint Bayesian method if standard PCA and linear projection are used because the highly uncertain features contaminate all the dimensions of the low dimensional space. Uncertainty needs to be taken into account throughout the whole processing pipeline to get good performance.

	Projection	
Training	Orthogonal	Probabilistic
PCA	20.2%	17.1%
UA-PPCA	18.8%	**14.7%**

Table 6.3: EER on MNIST with strong noise function of the dimensionality reduction method used for training (rows) and how the low dimensional projection is performed (columns)

6.3.2 Application to Face Verification

In this section, we show experiments on different face recognition datasets to demonstrate that uncertainty can contribute to cope with challenges like image resolution changes, occlusions and pose variations. We used the FRGC, PUT and MUCT datasets. FRGC is presented in Section 2.3.2, it includes face images of more than 500 identities acquired in controlled conditions, with little variations in pose. PUT [75] contains 9971 images from 100 persons. The subjects were asked to move their head in a specific way to get large pose variations. MUCT[76] also exhibits large pose variations. It contains 3755 images from 276 people. Pictures were simultaneously acquired with 5 cameras to obtain different views of each person. We report performance by looking at the False Negative Rate (FNR) at a False Positive Rate (FPR) of 0.1%.

6.3.2.1 Resolution Change

To illustrate how using uncertainty can contribute to deal with images of different resolutions, we have performed an experiment on FRGC. We cannot reuse the HOG-based feature vectors described in Section 2.3.2 because we need a representation for which a change in resolution clearly affects different features with different intensities.

The feature extraction process is composed of the following steps. We align the images using eyes location. The native inter-eye distance in FRGC images is approximately 80 pixels and during the alignment process the images are rescaled so that every image has an inter-eye distance of 64 pixels. We call those images *high resolution* images (HR). The feature vectors are composed of magnitudes of Gabor filters' response sampled on a regular grid (see [77], Section 4.4 for more information). We use 4 scales and 8 orientations and the resolution of the grid is specific to each scale (finer mesh at finer scales). The resulting feature vectors are 14216-dimensional. For all the experi-

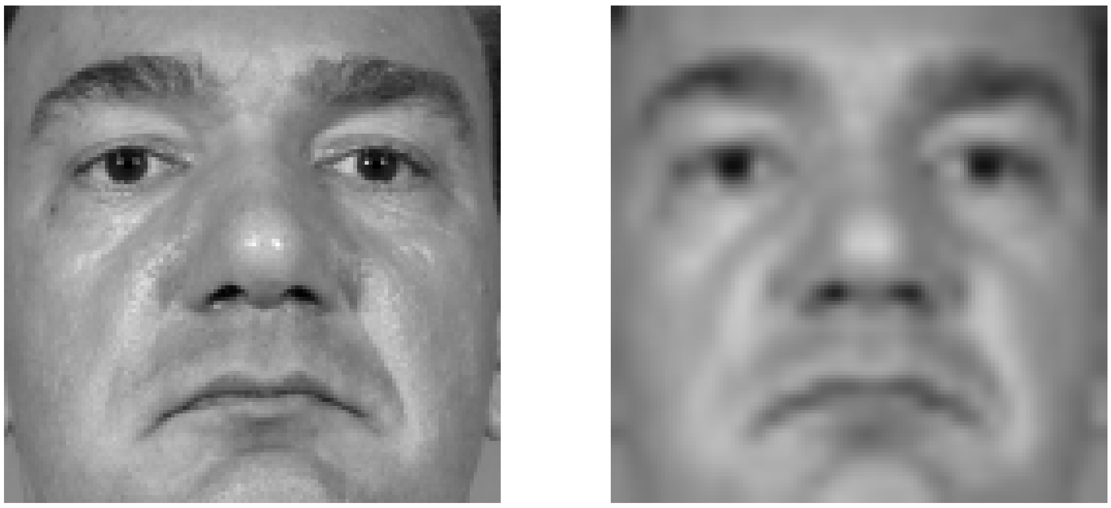


Figure 6.3: High resolution (left) and low resolution (right) versions of an FRGC image

ments with FRGC we have arbitrarily set the dimensionality of the space after reduction to 300. For other methods we compare ours to, standard PCA is used.

We created a low resolution (LR) version of each image by scaling it down by a factor 4 and then up by the same factor (using Lanczos resampling) so that they have the same size as the HR images. Figure 6.3 shows the two versions of an image.

The loss of resolution mostly affects the high frequency filters. It makes them noisier but also shrinks their distribution as shown in Figure 6.4. To cope with this issue we post-process each feature vector depending on the resolution of the image. First, we center each HR (resp. LR) feature vector around their HR (resp. LR) means. Second, we scale each component of LR feature vectors by a parameter so that the variance of the scaled component after post-processing is equal to the sum of the variance of this component in HR feature vectors plus the variance of the noise. The noise variance is estimated by the formula $\mathbb{E}\left[(x_{HR} - x_{LR})^2\right]$ on a dataset including for each image the HR and LR versions. The mean feature vectors and the factors have been computed once and for all on a special training dataset, hence they are used to post-process all the feature vectors involved in the training and the tests of the experiments presented in this section.

We now demonstrate the effectiveness of the proposed method to deal with scenarios where the training and tests are performed on images of different resolutions. To this end, we have performed three experiments which differ in the images used for training. The training of the first experiment is performed with the HR images, that of

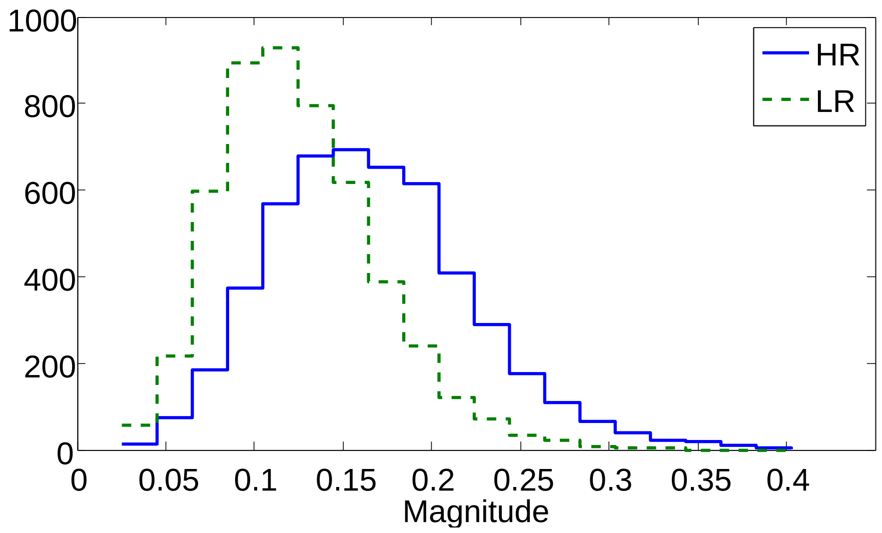


Figure 6.4: Histograms of the magnitude of a high frequency Gabor filter response on LR and HR images

		Methods			
Training	Test	UA-LR	JB	ITML	HL+FR
HR	HR	**2.5%**	**2.5%**	4.1%	**2.5%**
	LR	**4.1%**	6.3%	8.4%	6.7%
LR	HR	**3.0%**	3.2%	5.3%	3.8%
	LR	**3.0%**	4.2%	6.6%	4.2%
Mix	HR	**2.6%**	2.7%	6.8%	2.7%
	LR	**3.2%**	4.6%	7.5%	4.2%

Table 6.4: FNR at FPR=0.1% on FRGC depending on the training set and test set resolutions

the second with the LR images and that of the last experiment with a random mix of 50% of HR images and 50% of LR images. For each experiment we have evaluated the performance of all the methods on a test set of HR images and a test set of LR images. The results of the proposed method (UA-LR), Joint Bayesian [17], ITML [12] and HL+FR are presented in Table 6.4. UA-LR performs well in all configurations and it is worth noticing that, thanks to the use of the uncertainty, it is more robust than other methods. The benefit of using uncertainty is most visible when training on HR images because other methods tend to learn that the high-frequency Gabor filters are the most discriminative whereas these features are very noisy when the test set is composed of LR images.

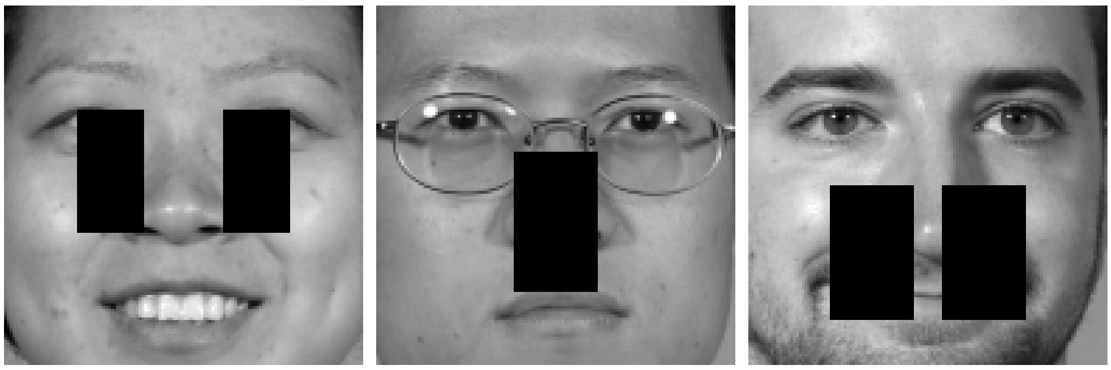


Figure 6.5: Examples of occluded faces

6.3.2.2 Dealing with Occlusion

Occlusion is one of the main challenges in face recognition and uncertainty offers a framework to handle it. In this experiment we use the non occluded HR images of FRGC described in the previous section for training. We have artificially created occluded test images by drawing random masks over the original images. The mask of each image is composed of two possibly overlapping rectangles which are symmetrical with respect to the vertical axis. We use symmetric masks because otherwise it would be too easy to recover the occluded part using the natural symmetry of faces. Figure 6.5 shows some examples of occluded faces. The masks on images are transformed into masks on feature vectors by considering that a feature is occluded if more than 5% of the energy of the corresponding filter is in an occluded area.

Similarity functions can only compare feature vectors of a fixed specific size, therefore we need to fill the occluded features. We use a standard missing data imputation scheme based on the conditional probability of the hidden data given the visible ones for normally distributed data. We can consider without loss of generality that, up to a feature reordering, all the occluded features are at the beginning of the feature vector. We use the formula of conditional multivariate normal random variables to compute the mean $o|_{v=a}$ and the covariance $S_o|_{v=a}$ of the filling pattern given the visible features v:

$$o|_{v=a} = \mu_o + C_{o,v}C_{v,v}^{-1}(\mu_v - a) \tag{6.36}$$

$$S_o|_{v=a} = C_o + C_{o,v}C_{v,v}^{-1}C_{v,o} \tag{6.37}$$

where μ_o and μ_v are respectively the mean of the occluded and visible features and C

	Methods			
	UA-LR	JB	ITML	HL+FR
Standard	**2.5%**	**2.5%**	4.1%	**2.5%**
Occluded	**8.0%**	9.8%	12.5%	11.9%

Table 6.5: Impact of occlusion on the FNR at FPR=0.1% on FRGC

is the covariance matrix of the features which has the following structure:

$$C = \begin{bmatrix} C_{o,o} & C_{o,v} \\ C_{v,o} & C_{v,v} \end{bmatrix}. \tag{6.38}$$

μ_o, μ_v and C are computed on the training set which is not occluded.

We provide to all methods the feature vectors where the occlusions have been filled with $o|_{v=a}$. $diag\left(S_o|_{v=a}\right)$ is used by UA-LR as uncertainty matrix and is ignored by other methods.

As in the previous section, UA-LR exhibits similar performance to Joint Bayesian and HL+FR on the original images but it outperforms them on the occluded images thanks to the use of uncertainty (see Table 6.5).

6.3.2.3 Robustness to Pose Variation

Robustness to pose variation is an important challenge for face recognition algorithms. A popular approach is to cancel most of the impact of pose variation with the help of a 3D morphable model. Synthetic frontal views are generated from non-frontal images and those synthetic images are used for comparison instead of the original ones. This process is called face frontalization. In our experiments, we use a method similar to that described in [78] and use the Gabor-based feature vectors described in Section 6.3.2.1. Creating frontal views from non-frontal images is a difficult task and artifacts might appear on generated images, especially in portions of frontalized images which correspond to areas poorly visible in the original non-frontal views. In this section, we show that performance is improved if the most affected areas are disregarded by the similarity function.

The pose of the face in a given image is estimated during the 3D morphable model fitting process. We automatically choose the mask of pixels to ignore among a set of

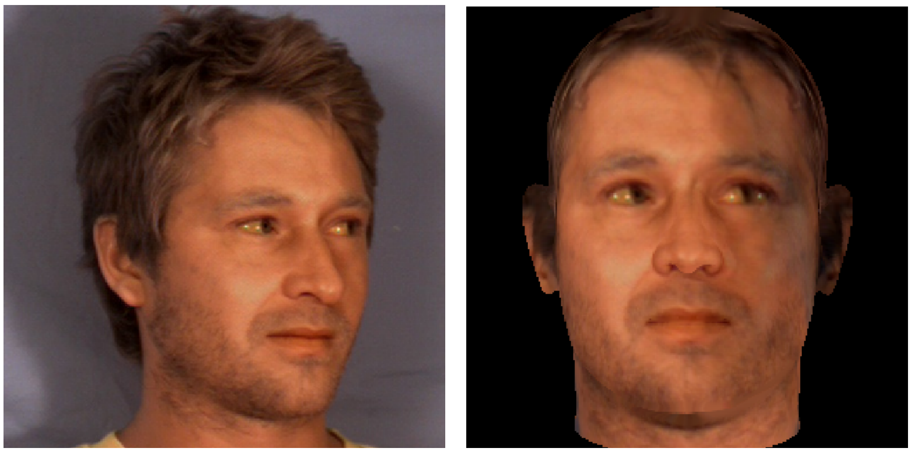


Figure 6.6: Original (left) and frontalized version (right) of an image from MUCT

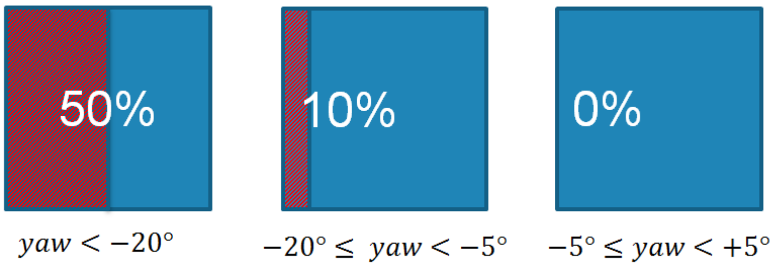


Figure 6.7: Masks associated with 3 of the 5 bins of yaw angle. The proportions of discarded pixels (hatched areas) are written in white.

predefined masks, function of the yaw angle estimated. Yaw angles are discretized into 5 bins: $yaw < -20°$, $-20° \leq yaw < -5°$, $-5° \leq yaw < +5°$, $+5° \leq yaw < +20°$ and $+20° \leq yaw$. Each bin is associated with a mask of pixels to ignore (see Figure 6.7). The discarded pixels are those which should be ignored during the comparison process because they are poorly visible on the original non-frontal image. These masks are transformed into uncertainty matrices on the feature vectors and are provided to our method exactly as explained in Section 6.3.2.2 for random occlusions.

We use the FRGC images to learn the parameters of our model (μ, W, S_μ and S_w) and test our similarity function on PUT and MUCT, two face datasets with large pose variations. We compare our method to standard PCA + Joint Bayesian. We may first note that error rates on databases with pose variations are not much higher than those we reported on FRGC in the previous section. This is due to the frontalization scheme used in this section; FNR are much higher if comparisons are performed on the original images. Second, we observe that using an uncertainty-aware similarity function leads to a notable improvement in performance on both databases despite the simple and

Table 6.6: Robustness to pose variations. FNR at FPR=0.1% on two databases with large variations in pose.

	Methods	
Database	UA-LR	JB
PUT	**2.7%**	3.1%
MUCT	**3.4%**	3.6%

coarse correspondence between yaw angles and pixel masks we use (see Table 6.6).

6.4 Conclusion

In this chapter, we have introduced a novel similarity learning method which, unlike previous approaches, can take advantage of the uncertainty information made available by the feature extraction process. The two stages of our method are based on probabilistic models and we provide EM algorithms to estimate their parameters.

Our experimental results indicate that it is beneficial to explicitly account for uncertainty information in similarity function learning. We demonstrate the effectiveness of our method on various challenging tasks such as performing face verification with images of various resolutions, pose variations or occlusions. However, our method has two limitations. First, it relies on Gaussian assumptions which might be over-simplistic for some data. Second, our algorithm only accepts variance as uncertainty information. Enabling it to deal with a broader class of auxiliary values would make the method more practical. For example, we would provide whether an image is high or low resolution as input and the algorithm would determine how this information should impact the similarity function. Overcoming those limitations are interesting challenges for future research.

Final Remarks and Future Work

The learning of similarity functions plays a central role in many pattern recognition tasks. Throughout this thesis, we have explored this topic and proposed novel methods ; this research has led to the publication of two conference papers [48, 71]. The work presented here has been carried out in collaboration with Safran Identity & Security which is a wold leader in biometric identification systems. This partnership has contributed to shape the course of this PhD. Many of the problems we have addressed in this thesis come from industrial needs and some algorithms we developed are now integrated into products. This partnership is also the reason why many illustrative examples are drawn from biometrics and, more specifically, from face recognition.

The development of a similarity function learning method requires the design of three important components:

1. An appropriate type of function must be chosen. In this thesis we have worked with linear metrics (Chapter 3), local metrics (Chapter 4), deep neural networks (Chapter 5) and log-likelihood functions (Chapter 6).
2. Using a well adapted regularizer also contributes to obtain good performance. We have presented a new regularizer for linear and local metrics in Section 3.3. As indicated in Section 6.2.3.3, considering the data to be uncertain is also a way to prevent over-fitting.
3. The training procedure for similarity function learning is more complicated than for standard regression or classification tasks. One reason is that training pairs are often not provided but have to be constructed from a multi-class dataset. The pair selection process has a strong impact on the overall performance. We have proposed several methods to create training pairs from labeled data in Section 3.1.5 and Section 5.2.

In many applications, scores output by similarity functions are thresholded to take a decision. The trade-off between false positive and false negative rates depends on this threshold, and hence the choice of the operating point is application-dependent: the focus can be placed on having either a low false positive rate (e.g. access control) or a low false negative rate (for example, forensic systems assisting police investigators finding suspects). To obtain good performance at low false negative rates, the similarity function needs to be robust to large intraclass variations. On the other hand, to be effective at low false positive rates, the similarity function should perform well on pairs of data points which are close in the original feature space. To improve performance at low false positive rates in heterogeneous datasets, we have introduced in Chapter 4 a novel local metric learning method called LMLML. Instead of using a single Mahalanobis distance everywhere in the feature space, LMLML combines several linear metrics into an adaptive similarity function which is able to handle local specificities. The objective function optimized during training also has an impact on the operating point at which performance is optimized. For methods which use pairs of feature vectors, adapting the pair selection strategy is one way to loosely focus on an operating point of interest. We have presented different methods of pairs selection in Section 3.1.5 and 5.2. However, to the best of our knowledge, there is no systematic approach to design objective function targeting specific operating points.

In Chapter 6, we have proposed the Uncertainty-Aware Likelihood Ratio, the first similarity learning method for data with uncertainty. This similarity function takes specific uncertainty information on the feature vectors as additional input. It is based on a generative model of the data which extends that proposed in [17]. Our method illustrates the potential of such approaches but there is still much to be done on this topic. Future work could address the two limitations of our approach. Namely, that it relies on strong Gaussian assumptions which might be too simple for some data and that it can only deal with a specific type of confidence information. It would also be valuable to output the confidence in a similarity score without auxiliary information on the feature vectors to compare. A similarity function is trained on a specific dataset. The similarity scores output for data pairs which are very different from those of the training set might be unreliable. To illustrate this, imagine a face verification system trained only on Asian faces. It might give erroneous similarity scores for people from other ethnicity, for example falsely considering that all Caucasians look the same. In addition to the similarity score, it would be useful to output a confidence value to modulate how the similarity is used in the system. One possible solution would be to combine the similarity function with a generative model of the training data to associate a similarity score and a confidence value to any data pair depending on how close to the training set the feature vectors are.

When similarity functions are used for nearest neighbors classification, the absolute value of the similarity scores do not matter, only the ranking does. On the other hand, many applications make a decision by comparing similarity scores to a predefined threshold. In this case, a correspondence between thresholds and performance measures has to be established. For example, when a face verification system verifies that a badge holder is its rightful owner to give access to a building, the false positive rate corresponds to the probability that a stolen badge could be used successfully. It is therefore mandatory to be able to determine accurately the threshold which leads to a targeted false positive rate. Moreover, the false positive rate corresponding to this threshold should be the same for all individuals. We want to avoid having badges being easier to fraud with than others. This problem has been formalized in [79] and is known in the biometric community as the Doddington zoo effect. Whereas the phenomenon has been extensively analyzed, there exists no systematic approach to build similarity functions with a stable correspondence between scores and false positive rates. Finding such methods would facilitate the integration of similarity functions in many operational systems.

During the last couple of years, the emergence of Deep Learning has changed the way many machine learning problems are addressed. We have seen in Chapter 5 that Neural Networks can also be successfully applied to similarity function learning. In this work, we have restricted ourselves to taking feature vectors as input and not the raw data in order to be able compare the results with the other similarity learning methods studied in this thesis. In practice, Deep Learning has blurred the distinction between feature extraction and decision making methods. Previously, feature representations were mostly handcrafted and only the decision stage was learned. With Deep Learning, the raw data such as images or speech signals is progressively transformed into increasingly discriminative features which eventually lead to a decision. However, most of the future research directions proposed above remain valid. Designing objective functions which target specific operating points and guarantee that no sample is prone to false positive errors and finding similarity functions which output confidence values are interesting challenges for Deep Learning-based similarity function learning methods.

Transfer learning is an interesting concept in which a model learned on a given dataset is adapted to a different type of data for which only few training samples are available. More recently, a related paradigm called Learning Using Privileged Information (LUPI) [80] has been introduced. It proposes to take advantage of additional information during training to mimic the interaction between a teacher and his student. In [80], the authors derive a method for classification called SVM+, it would be interesting to find how this paradigm could be applied to similarity function learning.

Appendices

Appendix A

Calculations for Uncertainty-Aware Likelihood Ratio

A.1 Calculation of $P(\mu_c|X_c,\bar{\Psi})$

The parameters of the distribution $P(\mu_c|X_c,\bar{\Psi})$ are used in the update formulas of S_μ (6.30) and S_w (6.35). In this section we show how we obtain them. We apply in sequence the Bayes theorem, the rule about affine transformation of normal random variables and that on Gaussian convolution to obtain:

$$
\begin{aligned}
P(\mu_c|X_c,\bar{\Psi}) &\propto P(\mu_c|\bar{S_\mu})P(X_c|\mu_c,\bar{S_w}) && \text{(A.1)}\\
&\propto \mathcal{N}(\mu_c|0,\bar{S_\mu})\prod_{i=1}^{m_c}\int \mathcal{N}(w_{c,i}|0,\bar{S_w})\mathcal{N}(x_{c,i}-w_{c,i}|\mu_c,S_{c,i})\,\mathrm{d}w_{c,i} && \text{(A.2)}\\
&\propto \mathcal{N}(\mu_c|0,\bar{S_\mu})\prod_{i=1}^{m_c}\mathcal{N}(x_{c,i}|\mu_c,\bar{S_w}+S_{c,i}) && \text{(A.3)}\\
&\propto \exp\left(-\frac{1}{2}\Big(\sum_{i=1}^{m_c}(\mu_c-x_{c,i})^\top\left(\bar{S_w}+S_{c,i}\right)^{-1}(\mu_c-x_{c,i})+\mu_c^\top\bar{S_\mu}^{-1}\mu_c\Big)\right) && \text{(A.4)}\\
&\propto \exp\left(-\frac{1}{2}(\mu_c-b_{\mu_c})^\top T_{\mu_c}^{-1}(\mu_c-b_{\mu_c})\right) && \text{(A.5)}
\end{aligned}
$$

$$
\text{where}\quad T_{\mu_c}=\left(\bar{S_\mu}^{-1}+\sum_{i=1}^{m_c}\left(\bar{S_w}+S_{c,i}\right)^{-1}\right)^{-1} \qquad \text{(A.6)}
$$

$$
\text{and}\quad b_{\mu_c}=T_{\mu_c}\sum_{i=1}^{m_c}\left(\bar{S_w}+S_{c,i}\right)^{-1}x_{c,i}. \qquad \text{(A.7)}
$$

$P(\mu_c|X_c,\bar{\Psi})$ is a probability distribution so (A.5) implies that $P(\mu_c|X_c,\bar{\Psi}) = \mathcal{N}(\mu_c|b_{\mu_c},T_{\mu_c})$.

A.2 Calculation of $P(w_{c,i}|X_c,\bar{\Psi})$

The update formula of S_w (6.35) depends on $P(w_{c,i}|X_c,\bar{\Psi})$. We detail here the calculations leading to the expression of its parameters. We introduce the variable μ_c that we marginalize out to obtain

$$P(w_{c,i}|X_c,\bar{\Psi}) = \int P(\mu_c|X_c,\bar{S_\mu})P(w_{c,i}|x_{c,i},\mu_c,\bar{S_w})\,\mathrm{d}\mu_c \tag{A.8}$$

We use the Bayes theorem and the rule on affine transformation of normal random variables and on the product of normal distributions to get the expression of $P(w_{c,i}|x_{c,i},\mu_c,\bar{S_w})$:

$$\begin{aligned}
P(w_{c,i}|x_{c,i},\mu_c,\bar{S_w}) &\propto \mathcal{N}(w_{c,i}|0,\bar{S_w})\mathcal{N}(x_{c,i}-\mu_c|w_{c,i},S_{c,i}) && \text{(A.9)}\\
&\propto \mathcal{N}(w_{c,i}|0,\bar{S_w})\mathcal{N}(w_{c,i}|x_{c,i}-\mu_c,S_{c,i}) && \text{(A.10)}\\
&\propto \mathcal{N}(w_{c,i}|R_{c,i}S_{c,i}^{-1}(x_{c,i}-\mu_c),R_{c,i}) && \text{(A.11)}\\
\text{where}\quad R_{c,i} &= \left(S_{c,i}^{-1}+\bar{S_w}^{-1}\right)^{-1}. && \text{(A.12)}
\end{aligned}$$

Moreover, as $P(w_{c,i}|x_{c,i},\mu_c,\bar{S_w})$ is a probability distribution, it is actually equal to $\mathcal{N}(w_{c,i}|R_{c,i}S_{c,i}^{-1}(x_{c,i}-\mu_c),R_{c,i})$. Applying again the rule of affine transformation of normal random variables in conjunction with that of the convolution of two normal distributions we obtain:

$$\begin{aligned}
P(w_{c,i}|X_c,\bar{\Psi}) &= \int \mathcal{N}(\mu_c|b_{\mu_c},T_{\mu_c})\mathcal{N}(w_{c,i}|R_{c,i}S_{c,i}^{-1}(x_{c,i}-\mu_c),R_{c,i})\,\mathrm{d}\mu_c && \text{(A.13)}\\
&= \int \mathcal{N}(\mu_c|b_{\mu_c},T_{\mu_c})\mathcal{N}(-S_{c,i}R_{c,i}^{-1}w_{c,i}-\mu_c|-x_{c,i},S_{c,i}R_{c,i}^{-1}S_{c,i})\,\mathrm{d}\mu_c && \text{(A.14)}\\
&= \mathcal{N}(-S_{c,i}R_{c,i}^{-1}w_{c,i}|b_{\mu_c}-x_{c,i},T_{\mu_c}+S_{c,i}R_{c,i}^{-1}S_{c,i}) && \text{(A.15)}\\
&= \mathcal{N}(w_{c,i}|b_{w_{c,i}},T_{w_{c,i}}) && \text{(A.16)}\\
\text{where}\quad T_{w_{c,i}} &= R_{c,i}S_{c,i}^{-1}T_{\mu_c}S_{c,i}^{-1}R_{c,i}+R_{c,i} && \text{(A.17)}\\
\text{and}\quad b_{w_{c,i}} &= R_{c,i}S_{c,i}^{-1}\left(x_{c,i}-b_{\mu_c}\right). && \text{(A.18)}
\end{aligned}$$

A.3 Maximization of $R(\Psi,\bar{\Psi})$ with respect to S_w

To maximize $R(\Psi,\bar{\Psi})$ (6.23) with respect to S_w, we compute its gradient and find the value of the matrix S_w which cancels it.

$$\frac{\partial R(\Psi,\bar{\Psi})}{\partial S_w} = \sum_{c=1}^{C} \int P(Z_c|X_c,\bar{\Psi}) \sum_{i=1}^{m_c} \frac{\partial \log P(w_{c,i}|S_w)}{\partial S_w} \mathrm{d}Z_c \tag{A.19}$$

$$= \sum_{c=1}^{C} \int \cdots \int P(\mu_c|X_c,\bar{\Psi}) \prod_{j=1}^{m_c} P(w_{c,j}|x_{c,j},\mu_c,\bar{\Psi}) \sum_{i=1}^{m_c} \frac{\partial \log P(w_{c,i}|S_w)}{\partial S_w} \prod_{j=1}^{m_c} \mathrm{d}w_{c,j}\,\mathrm{d}\mu_c. \tag{A.20}$$

$P(w_{c,i}|S_w)$ does not depend on the latent variables $w_{c,j}$ for $j \neq i$, it can therefore be taken out of the integrals. Moreover, all the integrals over $w_{c,j}$ are equal to 1 as they are integrals of random variables over their entire domain of definition. This leads to the following simplification:

$$\frac{\partial R(\Psi,\bar{\Psi})}{\partial S_w} = \sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}} \iint P(\mu_c|X_c,\bar{S_\mu}) P(w_{c,i}|x_{c,i},\mu_c,\bar{S_w})\,\mathrm{d}\mu_c \frac{\partial \log P(w_{c,i}|S_w)}{\partial S_w} \mathrm{d}w_{c,i} \tag{A.21}$$

$$= \sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}} \int P(w_{c,i}|X_c,\bar{\Psi}) \frac{\partial \log P(w_{c,i}|S_w)}{\partial S_w} \mathrm{d}w_{c,i} \tag{A.22}$$

$$\propto S_w^{-1} \sum_{c=1}^{C} m_c - S_w^{-1} \sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}} \int P(w_{c,i}|X_c,\bar{\Psi}) w_{c,i} w_{c,i}^{\top} \mathrm{d}w_{c,i} S_w^{-1} \tag{A.23}$$

$$\propto S_w^{-1} \sum_{c=1}^{C} m_c - S_w^{-1} \sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}} \left(T_{w_{c,i}} + b_{w_{c,i}} b_{w_{c,i}}^{\top} \right) S_w^{-1}. \tag{A.24}$$

We have therefore the following closed-form solution for the update formula:

$$S_w = \frac{1}{\sum_{c=1}^{C} m_c} \sum_{\substack{c=1,\\ i=1}}^{\substack{C,\\ m_c}} \left(T_{w_{c,i}} + b_{w_{c,i}} b_{w_{c,i}}^{\top} \right). \tag{A.25}$$

Bibliography

[1] Prasanta Chandra Mahalanobis. On the generalised distance in statistics. *Proceedings of the National Institute of Sciences of India*, 2(1):49–55, 1936.

[2] Andrew O. Hatch, Sachin Kajarekar, and Andreas Stolcke. Within-class covariance normalization for svm-based speaker recognition. In *ICSLP*, pages 1471–1474, 2006.

[3] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, 1973.

[4] Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 711–720, 1997.

[5] Martin Köstinger, Martin Hirzer, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Large scale metric learning from equivalence constraints. In *CVPR*, pages 2288–2295, 2012.

[6] Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *NIPS*, pages 505–512, 2003.

[7] Kilian Q. Weinberger and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

[8] Shibin Parameswaran and Kilian Q. Weinberger. Large margin multi-task metric learning. In *NIPS*, pages 1867–1875, 2010.

[9] Dor Kedem, Stephen Tyree, Fei Sha, Gert R. Lanckriet, and Kilian Q. Weinberger. Non-linear metric learning. In *NIPS*, pages 2573–2581, 2012.

[10] Kilian Q. Weinberger and Lawrence K. Saul. Fast solvers and efficient implementations for distance metric learning. In *ICML*, pages 1160–1167, 2008.

[11] Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. Is that you? metric learning approaches for face identification. In *ICCV*, pages 498–505, 2009.

[12] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *ICML*, pages 209–216, 2007.

[13] Rong Jin, Shijun Wang, and Yang Zhou. Regularized distance metric learning: Theory and algorithm. In *NIPS*, pages 862–870, 2009.

[14] Chunhua Shen, Junae Kim, Lei Wang, and Anton van den Hengel. Positive semidefinite metric learning with boosting. In *NIPS*, pages 1651–1659, 2009.

[15] Andreas Maurer. Learning similarity with operator-valued large-margin classifiers. *Journal of Machine Learning Research*, 9:1049–1082, 2008.

[16] Hieu V. Nguyen and Li Bai. Cosine similarity metric learning for face verification. In *ACCV*, pages 709–720, 2011.

[17] Dong Chen, Xudong Cao, Liwei Wang, Gang Wen, and Jian Sun. Bayesian face revisited: a joint formulation. In *ECCV*, 2012.

[18] Qiong Cao, Yiming Ying, and Peng Li. Similarity metric learning for face recognition. In *ICCV*, 2013.

[19] Zhen Li, Liangliang Cao, Shiyu Chang, John R. Smith, and Thomas S. Huang. Beyond mahalanobis distance: Learning second-order discriminant function for people verification. In *CVPR Workshops*, pages 45–50, 2012.

[20] Bernhard Scholkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA, 2001.

[21] Sebastian Mika, Gunnar Rätsch, Jason Weston, Bernhard Schölkopf, and Klaus-Robert Müller. Fisher discriminant analysis with kernels. In *Neural Networks for Signal Processing*, pages 41–48, 1999.

[22] Jane Bromley, Isabelle Guyon, Yann Lecun, Eduard Säckinger, and Roopak Shah. Signature verification using a ”siamese” time delay neural network. In *NIPS*, pages 737–744, 1994.

[23] Sumit Chopra, Raia Hadsell, and Yann Lecun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, pages 539–546, 2005.

[24] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[25] Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Discriminative deep metric learning for face verification in the wild. In *CVPR*, pages 1875–1882, 2014.

[26] Xinyuan Cai, Chunheng Wang, Baihua Xiao, Xue Chen, and Ji Zhou. Deep nonlinear metric learning with independent subspace analysis for face verification. In *ACM International Conference on Multimedia*, pages 749–752, 2012.

[27] Martin Köstinger, Peter M. Roth, and Horst Bischof. Synergy-based learning of facial identity. In *DAGM*, volume 7476 of *Lecture Notes in Computer Science*, pages 195–204, 2012.

[28] Yung-Kyun Noh, Byoung-Tak Zhang, and Daniel D. Lee. Generative local metric learning for nearest neighbor classification. In *NIPS*, pages 1822–1830, 2010.

[29] Jun Wang, Alexandros Kalousis, and Adam Woznica. Parametric local metric learning for nearest neighbor classification. In *NIPS*, pages 1610–1618, 2012.

[30] Shreyas Saxena and Jakob Verbeek. Coordinated local metric learning. In *ICCV ChaLearn Looking at People workshop*, 2015.

[31] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, 2007.

[32] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *NIPS*, 2014.

[33] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):971–987, 2002.

[34] Jonathon P. Phillips, Patrick J. Flynn, Todd Scruggs, Kevin W. Bowyer, Jin Chang, Kevin Hoffman, Joe Marques, Jaesik Min, and William Worek. Overview of the face recognition grand challenge. In *CVPR*, pages 947–954, 2005.

[35] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.

[36] Andrea Vedaldi and Brian Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.

[37] Andreas Maurer. Generalization bounds for subspace selection and hyperbolic pca. In *Subspace, Latent Structure and Feature Selection*, pages 185–197, 2006.

[38] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, pages 148–156, 1996.

[39] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z. Li. Learning face representation from scratch. *CoRR*, abs/1411.7923, 2014.

[40] Jack Kiefer. Sequential minimax search for a maximum. *Proceedings of The American Mathematical Society*, 4:502–502, 1953.

[41] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45:503–528, 1989.

[42] Magnus R. Hestenes and Eduard Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[43] Léon Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*. EC2, 1991.

[44] Léon Bottou. In Grgoire Montavon, Genevieve B. Orr, and Klaus-Robert Mller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, pages 421–436.

[45] Roberto Battiti. Accelerated backpropagation learning: Two optimization methods. *Complex Systems*, 3(4):331–342, 1989.

[46] Andreas Maurer. Learning to compare using operator-valued large-margin classifiers. In *NIPS, LTCE Workshop*, 2006.

[47] Lubor Ladickỳ and Philip H. S. Torr. Locally linear support vector machines. In *ICML*, pages 985–992, 2011.

[48] Julien Bohné, Yiming Ying, Stéphane Gentric, and Massimiliano Pontil. Large margin local metric learning. In *ECCV*, 2014.

[49] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[50] Karen Simonyan, Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Fisher vector faces in the wild. In *BMVC*, 2013.

[51] Lei Zhang and David Zhang. Evolutionary cost-sensitive extreme learning machine and subspace extension. *CoRR*, abs/1505.04373, 2015.

[52] Meina Kan, Shiguang Shan, Dong Xu, and Xilin Chen. Side-information based linear discriminant analysis for face recognition. In *BMVC*, pages 1–12, 2011.

[53] Yiming Ying and Peng Li. Distance metric learning with eigenvalue optimization. *Journal of Machine Learning Research*, 13:1–26, 2012.

[54] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[55] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

[56] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, pages 315–323, 2011.

[57] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, pages 1929–1958, 2014.

[58] Raia Hadsell, Sumit Chopra, and Yann Lecun. Dimensionality reduction by learning an invariant mapping. In *CVPR*, pages 1735–1742, 2006.

[59] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[60] Ian J. Goodfellow, David Warde-farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *ICML*, 2013.

[61] Jia-Ren Chang and Yong-Sheng Chen. Batch-normalized maxout network in network. *CoRR*, abs/1511.02583, 2015.

[62] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[63] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48, 2009.

[64] Stan Salvador and Philip Chan. Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, pages 561–580, 2007.

[65] Jinbo Bi and Tong Zhang. Support vector classification with input data uncertainty. In *NIPS*, pages 1651–1659, 2004.

[66] Pannagadatta K. Shivaswamy, Chiranjib Bhattacharyya, and Alexander J. Smola. Second order cone programming approaches for handling missing and uncertain data. *Journal of Machine Learning Research*, 7:1283–1314, 2006.

[67] Smith Tsang, Ben Kao, Kevin Y. Yip, Wai-Shing Ho, and Sau Dan Lee. Decision trees for uncertain data. *IEEE Transactions on Knowledge and Data Engineering*, 23:64–78, 2011.

[68] Jiangtao Ren, Sau Dan Lee, Xianlu Chen, Ben Kao, Reynold Cheng, and David Wai-Lok Cheung. Naive bayes classification of uncertain data. In *ICDM*, 2009.

[69] Graham Cormode and Andrew McGregor. Approximation algorithms for clustering uncertain data. In *PODS*, 2008.

[70] Hanz-Peter Kriegel and Martin Pfeifle. Hierarchical density-based clustering of uncertain data. In *ICDM*, 2005.

[71] Julien Bohné, Sylvain Colin, Stéphane Gentric, and Massimiliano Pontil. Similarity function learning with data uncertainty. In *ICPRAM*, 2016.

[72] Michael E. Tipping and Chris M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 61:611–622, 1999.

[73] Simon J.D. Prince and James H. Elder. Probabilistic linear discriminant analysis for inferences about identity. In *ICCV*, 2007.

[74] Christopher M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, page 108116, 1995.

[75] Andrzej Kasiński, Andrzej Florek, and Adam Schmidt. The put face database. *Image Processing & Communication*, 13/3:59–64, 2008.

[76] Stephen Milborrow, John Morkel, and Fred Nicolls. The muct landmarked face database. *Pattern Recognition Association of South Africa*, 2010. `http://www.milbo.org/muct`.

[77] Stan Z. Li and Anil K. Jain. *Handbook of Face Recognition 2nd ed.* Springer, 2011.

[78] Volker Blanz, Patrick Grother, Jonathon P. Phillips, and Thomas Vetter. Face recognition based on frontal views generated from non-frontal images. In *CVPR*, pages 454–461, 2005.

[79] George Doddington, Walter Liggett, Alvin Martin, Mark Przybocki, and Douglas Reynolds. Sheep, goats, lambs and wolves a statistical analysis of speaker performance in the nist 1998 speaker recognition evaluation. In *International Conference On Spoken Language Processing*, 1998.

[80] Vladimir Vapnik and Akshay Vashist. A new learning paradigm: Learning using privileged information. *Neural Networks*, pages 544–557, 2009.