UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

# Metric learning for efficient search with high-dimensional multi-modal data

Dissertation presented by
**Victor DEPLASSE**

for obtaining the Master's degree in
**Computer Science and Engineering**

Supervisor(s)
**Pierre DUPONT, Thomas PEEL**

Reader(s)
**Aymen CHERIF, John LEE**

Academic year 2016-2017

# Abstract

Many machine learning algorithms are based on the similarity or distance between objects. For these algorithms, metric learning is a useful preprocessing step to learn a task-specific metric. But until now, metric learning techniques have mainly focused on uni-modal data, while multi-modal data increasingly arise in real-world cases, such as multimedia applications. This master's thesis first explores if taking the multi-modal aspect of the data into account when learning a metric allows to increase the performance of a kNN classifier. For this, multiple ways to combine the modalities are experimented and some problems that can arise with iterative algorithms are tackled, for instance fixing the learning rate.

The state-of-the-art metric learning techniques for multi-modal data do not scale well with the size of the training set. This master's thesis explores some approaches to reduce the execution time of these algorithms. It proposes the MKPOE-LR algorithm to tackle the problem of efficiently learning an effective metric for large datasets. It also shows that some other approaches, such as limiting the number of distance constraints, perform well on the classification task at hand while having a significantly smaller execution time.

# Acknowledgements

To my supervisor, Prof. Pierre Dupont, thank you for your guidance, your constructive feedbacks during the year and for the relevant questions that challenged me to deepen my understanding. But also for the excellent courses on Machine Learning, which gave me the desire to do my master's thesis in this field.

To Dr. Thomas Peel from EURA NOVA, thank you for your time, for the comments during our weekly meetings and for helping me develop my work. Your advices and feedbacks were very valuable throughout my master's thesis.

To my parents, thank you for your encouragements and continued support since the beginning of my studies.

# Contents

# List of symbols

| | |
|---|---|
| $\mathbb{R}$ | Set of real numbers |
| $\mathbb{R}^d$ | Set of $d$-dimensional real-valued vectors |
| $\mathbb{R}^{n \times d}$ | Set of $n \times d$ real-valued matrices |
| $\mathcal{X} = \{x_i\}_{i=1}^n$ | Training set of $n$ examples |
| $n$ | Size of the training set |
| $d$ | Dimension of the original space |
| $\mathcal{S}$ | Set of similarity constraints |
| $\mathcal{D}$ | Set of dissimilarity constraints |
| $\mathcal{R}$ | Set of relative (triplet) constraints |
| $d(x_i, x_j)$ | Mahalanobis distance between $x_i$ and $x_j$ (in the sense of the metric learning litterature) |
| $\ell(\cdot)$ | Loss function |
| $m$ | Number of modalities |
| $p$ | Index of a modality (value in $\{1, \ldots, m\}$) |
| $S(x_i, x_j)$ | Similarity measure between $x_i$ and $x_j$ |
| $k(x_i, x_j)$ | Kernel function between $x_i$ and $x_j$ |
| $K$ | Kernel matrix |
| $K_i$ | Column $i$ of the kernel matrix $K$ |
| $\hat{K}$ | Kernel obtained from a combination of kernels |
| $I$ | Identity matrix |
| $M$ | Parameter matrix of the learned Mahalanobis distance, for linear metric learning techniques |
| $L$ | Linear transformation |
| $W$ | Parameter matrix of the learned Mahalanobis distance, for nonlinear metric learning techniques |
| $W^p$ | Parameter matrix of the learned Mahalanobis distance corresponding to the modality $p$ |

| | |
|---|---|
| $\Lambda$ | Diagonal matrix whose diagonal elements are the eigenvalues of $W$ |
| $V$ | Matrix whose columns are the eigenvectors of $W$ |
| $P$ | Random projection matrix |
| $Q$ | Matrix with the embeddings $g(x_i)$ (for all training point $x_i$) as columns |
| $M \succeq 0$ | A positive semi-definite matrix $M$ |
| $\mathrm{tr}(\cdot)$ | Trace of a matrix |
| $\|\cdot\|_2$ | Euclidean norm of a vector |
| $\|\cdot\|_{\mathcal{F}}$ | Frobenius norm of a matrix |
| $\|\cdot\|_{HS}$ | Hilbert-Schmidt norm of a matrix |
| $\xi$ | Slack variable |
| $\mu$ | Weight value |
| $g(x)$ | Nonlinear embedding function |
| $\phi(x)$ | Feature map |
| $\phi_i$ | Shortcut for $\phi(x_i)$ |
| $\alpha$ | Learning rate of the iterative methods; significance level for the statistical tests |
| $\lambda$ | Hyperparameter controlling the trade-off between regularization and loss minimization; scale hyperparameter for 4-sparse matrices |
| $t$ | Iteration number |
| $B$ | 4-sparse base matrix |
| $\gamma_B$ | Weight of a base matrix $B$ in the Frank-Wolfe algorithm |
| $\gamma$ | Hyperparameter of the RBF kernel |
| $r_i^j$ | Rank of algorithm $j$ on the dataset $i$ for the statistical tests |
| $F_F$ | Friedman statistic |
| $\mathcal{L}$ | Symmetric normalized Laplacian matrix |
| $\langle\cdot,\cdot\rangle_{\mathcal{F}}$ | Frobenius inner product |

# List of abbreviations

| | |
|---|---|
| **kNN** | k-nearest neighbors |
| **KPOE** | Kernel Partial Order Embedding |
| **LMNN** | Large Margin Nearest Neighbor |
| **MKPOE** | Multiple Kernel Partial Order Embedding |
| **MKPOE-LR** | Low-Rank Multiple Kernel Partial Order Embedding |
| **MSD** | Million Song Dataset |
| **OMDL** | Online Multi-modal Distance Learning |
| **OMDL-LR** | Low-Rank Online Multi-modal Distance Learning |
| **PSD** | Positive semi-definite |
| **RBF** | Radial basis function |
| **RBM** | Restricted Boltzmann machine |
| **SD** | Standard deviation |
| **tf-idf** | Term frequency - inverse document frequency |
| **Top-MAGD** | MSD Allmusic Top Genre Dataset |

# Introduction

The notion of distance or similarity between objects is used in many machine learning algorithms, such as the k-nearest neighbors (kNN) algorithm for classification or regression, the k-means algorithm for clustering, and kernel-based algorithms such as Support Vector Machine. Information retrieval systems also use similarity between objects, for example, to rank the results of a search query. Assuming that the objects are represented by a vector of numerical features, the metric (or distance function) generally used by those algorithms is the Euclidean distance. But, depending on the task or the data available, it is not always the best metric to use. This is where metric learning is necessary: it is a preprocessing step that consists in automatically learning a metric in a supervised way.

Until now, metric learning has mainly focused on uni-modal data (e.g. text only) but multi-modal data arise naturally in many real-world cases, where data come from multiple heterogeneous sources or are represented by different types of feature representations. For instance, a web page can contain multiple types of data, such as texts or photos; a song can be represented by audio features, metadata, its genre or lyrics. Taking multiple modalities into account could provide interesting results when computing the similarity between objects. However, the way these modalities should be combined is not obvious. The objective of this master's thesis is to propose a technique to address the metric learning task for high-dimensional multi-modal data. The proposed technique is assessed on a subset of the Million Song Dataset [5], a dataset containing audio features and metadata of one million songs.

## Contributions

This master's thesis shows that taking advantage of the multi-modal aspect of the data during a metric learning step allows to increase the accuracy of a kNN classifier for a genre recognition task, using data from the Million Song Dataset. Multiple ways to combine the modalities of the data are compared in order to confirm the effectiveness of Multiple Kernel Partial Order Embedding (MKPOE) [19], a state-of-the-art metric learning technique for multi-modal data. It also shows that this technique fails to learn a good distance measure on an artist recognition task because of the small number of available songs for each artist compared to the high dimensions of the input space.

MKPOE is based on a subgradient descent, but little information is provided on how to choose the learning rate in an efficient way. Another contribution of this master's thesis is to propose a technique for choosing the learning rate in such a way that the convergence is easily achieved, and to compare it to other well-known approaches.

Finally, as MKPOE is computationally expensive, a few approaches to reduce its execution time are explored. This is useful to scale the technique to large datasets.

## Outline

This master's thesis is organized as follows. The first chapter contains an overview of metric learning. Chapter 2 presents two state-of-the-art techniques for multi-modal metric learning. The first technique is close to the usual metric learning litterature while the other is based on neural networks. The third chapter describes the experimental setting used to assess the performance of the proposed technique. It contains a description of the datasets and the tasks on which the performance is assessed. It ends by an explanation of how the learning rate is chosen in the iterative algorithm used to solve the metric learning problem. In chapter 4, it is first studied if the multi-modal aspect of the data can be used to increase the accuracy of the two classification tasks at hand. Then multiple ways to combine the modalities are compared. This chapter also contains a description of some implementation details. In chapter 5, a few possibilities to reduce the execution time of MKPOE are explored in order to scale this technique to large datasets. The last part of this master's thesis concludes the research and discusses further possible research directions.

# Chapter 1

# Metric learning, an overview

Depending on the task, the similarity or distance between objects of a dataset could be different. Lets take a dataset of face images to illustrate this. A face recognition application, which aims at recognizing a person's identity, will consider pictures of the same person as similar. In this case, the similarity between two images should be based on features such as the color of the eyes, the color of the hair or the shape of the face. Another task could be to recognize the pose of a person. In this case, it is the pictures of persons in the same pose that should be considered similar. The similarity between two images should thus be based on another set of features, such as the number of visible ears. A third task could be to recognize facial expressions. For this task, the similarity should be based on features such as the position of the eyebrouws, mouth, etc. Many other tasks can be defined on this dataset, such as age or gender recognition.

The Figure 1.1 from [17] illustrates a dataset of face images. For the face recognition task, images in the columns should be considered similar, but in the second task images in the rows should be considered similar. So the metric will not be the same in both tasks. It is not trivial to determine an appropriate metric by hand, even for an expert in the domain of the task, because it requires to correctly weight and combine the features. Hopefully, metric learning allows to automatically learn a task-specific metric in a supervised way, by being provided with some side-information about the objects that are considered similar or not for the task at hand.

Figure 1.1: Example of a dataset of face images. Image taken from [17]

The side information needed to learn a task-specific metric is often of one of the following forms, where $x_i$, $x_j$ and $x_k$ are objects from the dataset:

- Must-link / cannot-link constraints:

$$\mathcal{S} = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be similar}\},$$
$$\mathcal{D} = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be dissimilar}\}$$

  This is the simplest form, where the metric learning algorithm is provided with pairs of similar ($\mathcal{S}$) or dissimilar ($\mathcal{D}$) data points.

- Triplet constraints, also called relative constraints:

$$\mathcal{R} = \{(x_i,x_j,x_k) \text{: } x_i \text{ should be more similar to } x_j \text{ than to } x_k\}$$

  In this case the metric is learned such that the distance between $x_i$ and $x_j$ is smaller than the distance between $x_i$ and $x_k$.

The first section of this chapter presents linear metric learning and the second section presents nonlinear metric learning. The content of this chapter is mainly based on surveys [3, 4, 17].

## 1.1 Linear metric learning

When the objects are represented as data points in a vector space $\mathbb{R}^d$, most linear metric learning algorithms learn a metric of the form:

$$d(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)} \tag{1.1}$$

where $x_i$ and $x_j$ are two data points in $\mathbb{R}^d$ and $M \in \mathbb{R}^{d \times d} \succeq 0$ is a parameter positive semi-definite (PSD) matrix. This form looks like the Mahalanobis distance:

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T \Sigma^{-1} (x_i - x_j)} \tag{1.2}$$

where $\Sigma$ is the covariance matrix of the data points. Because of this similarity, this form is often called a Mahalanobis distance by an abuse of terminolgy.

The goal of a metric learning algorithm is to find the matrix $M$. It's easy to see that $M$ should be PSD in order to have a positive value inside the square root. If $M = I$ (the identity matrix), then the distance in Equation (1.1) is the Euclidean distance.

Since $M$ is PSD, the Cholesky decomposition can be used to express $M$ as $L^T L$, where $L \in \mathbb{R}^{r \times d}$ and $r$ is the rank of $M$. With this decomposition, the Mahalanobis distance becomes:

$$\begin{aligned} d(x_i, x_j) &= \sqrt{(x_i - x_j)^T M (x_i - x_j)} \\ &= \sqrt{(x_i - x_j)^T L^T L (x_i - x_j)} \\ &= \sqrt{(Lx_i - Lx_j)^T (Lx_i - Lx_j)} \\ &= \|Lx_i - Lx_j\|_2 \end{aligned} \tag{1.3}$$

So learning a Mahalanobis distance can be seen as learning a linear transformation of the data, defined by the transformation matrix $L$, and using the Euclidean distance after mapping the data to the new space. Figure 1.2 illustrates metric learning applied to a face recognition task where the images are represented by points in a two-dimensional space. In this illustration, the learned metric can be seen as a linear transformation of the data in the plane such that the Euclidean distance between objects in the transformed space satisfies as many distance constraints as possible.

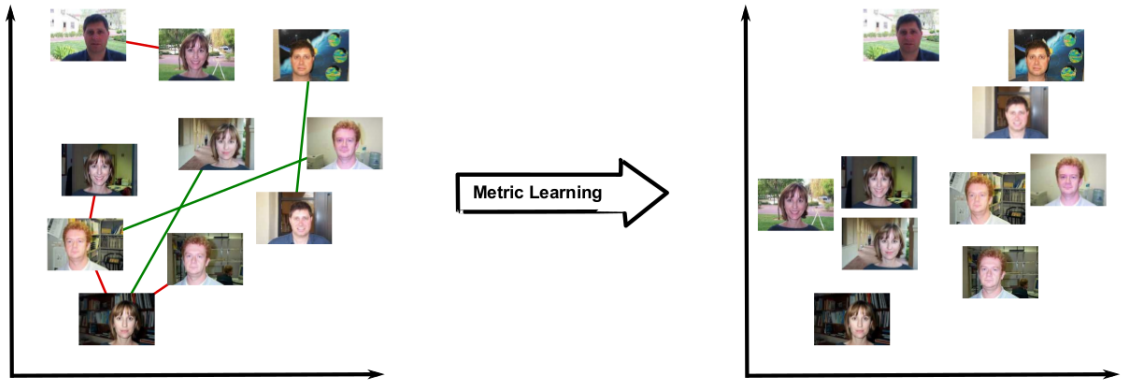



Figure 1.2: from [3]: Illustration of metric learning applied to a face recognition task. Images representing the same person are linked by must-link constraints (in green) and images of different persons are linked by cannot-link constraints (in red).

**Definition 1.** A *metric* (or *distance function* or simply *distance*) on a set $\mathcal{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that respects these 4 conditions $\forall x_i, x_j, x_k \in \mathcal{X}$:

1. $d(x_i, x_j) \geq 0$ (non-negativity)
2. $d(x_i, x_j) = 0 \Leftrightarrow x_i = x_j$ (identity of indiscernibles)
3. $d(x_i, x_j) = d(x_j, x_i)$ (symmetry)
4. $d(x_i, x_k) \leq d(x_i, x_j) + d(x_j, x_k)$ (triangle inequality)

$M$ is PSD so its rank $r$ can be lower than $d$. Since $L \in \mathbb{R}^{r \times d}$, if $r < d$, the linear transformation maps the data to a lower-dimensional space and some distinct points in $\mathbb{R}^d$ can be mapped to the same point in $\mathbb{R}^r$, so the distance between them is 0 after transformation. In this case, the second condition of a metric is not totally met and what is learned instead is a pseudo-metric.

### 1.1.1 Well-known early approach

One well-known early approach of metric learning is Large Margin Nearest Neighbors (LMNN) introduced by Weinberger et al. in 2006 [24]. Their idea was to design a metric learning algorithm to improve kNN classification. To do this, the constraints are generated from a labelled training set $\{(x_i, y_i)\}_{i=1}^n$. For each training example $x_i$, the $k$ nearest neighbors of the same class (called the target neighbors) should be closer than examples from another class (called the impostors). These $k$ target neighbors are selected using the Euclidean distance and two sets of distance constraints are constructed:

$$\begin{aligned}\mathcal{S}_{\text{lmnn}} &= \{(x_i, x_j) : y_i = y_j \text{ and } x_j \text{ belongs to the } k\text{-neighborhood of } x_i\},\\ \mathcal{R}_{\text{lmnn}} &= \{(x_i, x_j, x_k) : (x_i, x_j) \in \mathcal{S}_{\text{lmnn}}, y_i \neq y_k\}.\end{aligned}$$

The Mahalanobis distance is learned using the following convex minimization problem:

$$\begin{aligned}\min_{M,\xi} \quad & (1-\mu) \sum_{(x_i,x_j)\in\mathcal{S}_{\text{lmnn}}} d^2(x_i, x_j) + \mu \sum_{(x_i,x_j,x_k)\in\mathcal{R}_{\text{lmnn}}} \xi_{ijk} \\ \text{s.t.} \quad & d^2(x_i, x_k) - d^2(x_i, x_j) \geq 1 - \xi_{ijk}, \\ & \xi_{ijk} \geq 0 \qquad \forall (x_i, x_j, x_k) \in \mathcal{R}_{\text{lmnn}}, \\ & M \succeq 0\end{aligned} \tag{1.4}$$

where $\mu \in [0, 1]$ is a trade-off hyperparameter between bringing the target neighbors closer from $x_i$ and pushing away the impostors. The $\xi_{ijk}$ are slack variables to minimize, introduced because it may be infeasible to satisfy all the constraints with margin 1. Figure 1.3 illustrates LMNN for one training example for a $k$-neighborhood with $k = 3$.

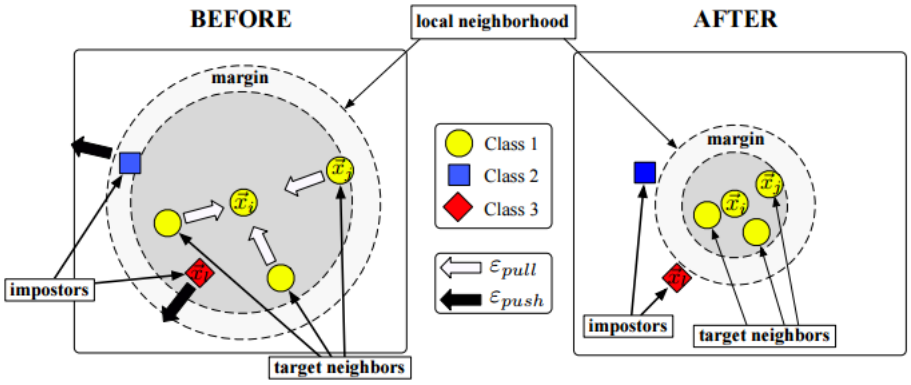



Figure 1.3: Illustration of LMNN (taken from [25]) for one training example before and after training for $k = 3$

### 1.1.2 Regularized approaches

In order to avoid overfitting, more recent metric learning approaches include a regularization term [4]. The optimization problem is formulated in the form:

$$\min_{M} \ \ell(M, \mathcal{S}, \mathcal{D}, \mathcal{R}) + \lambda R(M) \tag{1.5}$$

where $\ell(M, \mathcal{S}, \mathcal{D}, \mathcal{R})$ is a loss function that penalizes the violation of constraints from the sets $\mathcal{S}$, $\mathcal{D}$ or $\mathcal{R}$, $R(M)$ is a regularization term that penalizes complex models and $\lambda \geq 0$ is a hyperparameter controlling the trade-off between regularization and satisfying the constraints.

Schultz and Joachims [21] propose to learn a metric for classification using a set of relative constraints $\mathcal{R}$ by solving this soft-margin problem:

$$\begin{aligned} \min_{M, \xi} \quad & \sum_{(x_i, x_j, x_k) \in \mathcal{R}} \xi_{ijk} + \lambda \|M\|_{\mathcal{F}}^2 \\ \text{s.t.} \quad & d^2(x_i, x_k) - d^2(x_i, x_j) \geq 1 - \xi_{ijk}, \\ & \xi_{ijk} \geq 0 \qquad \forall (x_i, x_j, x_k) \in \mathcal{R}, \\ & M \succeq 0 \end{aligned} \tag{1.6}$$

where $\|M\|_{\mathcal{F}}^2$ is the squared Frobenius norm of $M$:

$$\|M\|_{\mathcal{F}}^2 = \sum_{i=1}^{d} \sum_{j=1}^{d} M_{ij}^2 \tag{1.7}$$

This is a simple regularizer that can be viewed as the squared $L_2$-norm regularizer for vectors applied to matrices.

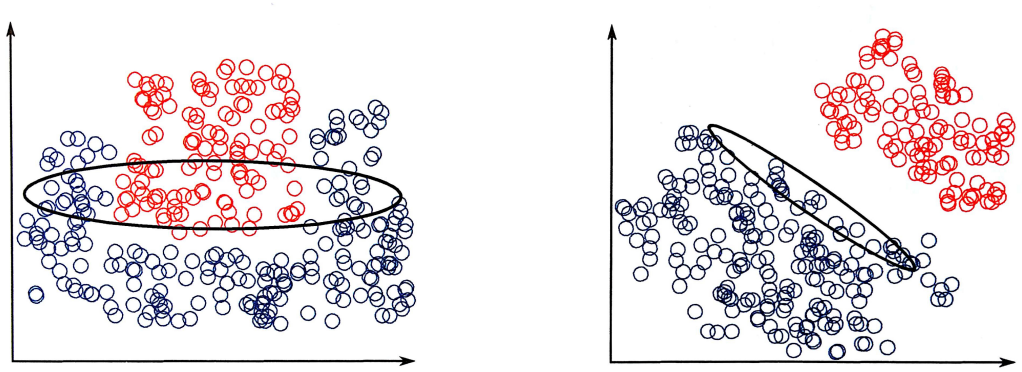
As in LMNN (problem (1.4)), this technique also tries to satisfy the triplet constraints in $\mathcal{R}$, but it doesn't have the second objective of minimizing the distances between similar points. It thus doesn't need a set $\mathcal{S}$ of similar points. Instead, it has an additional general regularization term that favors simple models. Minimizing the distance between similar points in LMNN can be

seen as favorizing simple models, but with the emphasis on the impact of similar points on the metric. While the approach of Schultz and Joachims uses a general regularization term which has a greater regularization power. Moreover, the influence of this second objective on the model in LMNN depends on the number of targets neighbors. If all the points of the same class are considered similar then LMNN minimizes the within-class distances. But the regularization term in the problem (1.6) is general to the full metric learned.

## 1.2 Nonlinear metric learning

Many datasets have a nonlinear structure that linear metric learning methods are unable to capture. For instance, when dealing with a classification task, a lot of datasets may not be linearly separable. A linear metric cannot perform well with such datasets. To justify this, a Mahalanobis distance can be represented in the original space by an ellipse (or an ellipsoid in high-dimensional spaces) representing the set of points equidistant from its center. An example of a non-linearly separable dataset is presented in Figure 1.4a. This figure shows that a linear metric cannot perform well when the data is not linearly separable. In such situations, nonlinear metrics should perform better.

In a nonlinear metric learning scheme, a (generally nonlinear) function $g$ that embeds the data points in a new space is learned, such that as many distance constraints as possible are satisfied in this embedding space when using the Euclidean distance between points.



(a) Linear metric

(b) Kernelized metric

Figure 1.4: Example (from [4]) of a non-linearly separable dataset where nonlinear metric learning is needed

Based on surveys on metric learning [3, 4, 17], two main nonlinear metric learning approaches exist: learning a linear metric in the feature space induced by a kernel (Figure 1.4b) or directly learning a nonlinear metric. Compared to linear metric learning, where a matrix of parameters is learned, directly learning a nonlinear metric is not easy. The kernelization of linear transformations has been successful in the metric learning litterature thanks to its ability to learn a matrix of parameters. Because it has the advantage to combine the expressiveness of a nonlinear metric and the efficiency of a linear metric learning technique, which typically optimizes a convex problem, the rest of this section discusses about the kernelization of linear methods.

As proposed in [4] and [19], let $\phi : \mathbb{R}^d \to \mathbb{R}^D$ be a (generally nonlinear) function that maps the data points to a (possibly infinite) $D$-dimensional feature space equipped with a kernel function $k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$. Let $\phi(x_i) = \phi_i$ for simplicity. The Mahalanobis distance is learned in

the feature space:

$$d(\phi_i, \phi_j) = \sqrt{(\phi_i - \phi_j)^T M (\phi_i - \phi_j)} = \sqrt{(\phi_i - \phi_j)^T L^T L (\phi_i - \phi_j)} \tag{1.8}$$

This can be seen as learning a linear transformation $L \in \mathbb{R}^{D' \times D}$ in the feature space induced by $\phi(x)$:

$$g(x) = L(\phi(x)) \tag{1.9}$$

So $g(x)$ embeds the data points in a finite-dimensional space $\mathbb{R}^{D'}$.

Let $\Phi = (\phi_1, ..., \phi_n)$ be a column-wise matrix representation of the training set (of size $n$) in the feature space obtained by $\phi(x)$. It may not be feasible to directly optimize $L$ because the space in which the data are projected by $\phi$ can be infinite-dimensional. Instead, the parameterization

$$L = N\Phi^T \tag{1.10}$$

can be used, with a matrix $N \in \mathbb{R}^{D' \times n}$, to transform the Mahalanobis distance in Equation (1.8):

$$\begin{aligned} d(\phi_i, \phi_j) &= \sqrt{(\phi_i - \phi_j)^T \Phi N^T N \Phi^T (\phi_i - \phi_j)} \\ &= \sqrt{(\Phi^T(\phi_i) - \Phi^T(\phi_j))^T N^T N (\Phi^T(\phi_i) - \Phi^T(\phi_j))} \\ &= \sqrt{(K_i - K_j)^T N^T N (K_i - K_j)} \\ &= \sqrt{(K_i - K_j)^T W (K_i - K_j)} \end{aligned} \tag{1.11}$$

where $W = N^T N \in \mathbb{R}^{n \times n}$ and $K_i = \Phi^T(\phi_i) = (k(x_i, x_1), \ldots, k(x_i, x_n))^T$ is the column vector obtained when computing the kernel between $x_i$ and each training point. Thanks to the kernel trick, the Mahalanobis distance can be computed by using the kernel function in Equation (1.11), without the need to project into the feature space induced by $\phi(x)$. The kernel trick is useful especially because there are many cases where the function $\phi(x)$ is unknown. Furthermore, the computation of the kernel is often cheaper than explicitly projecting the points into the feature space, which is often of higher dimension than the original space. It can also be noted that the implicit feature space may be infinite-dimensional. For instance, this is the case when using a radial basis function (RBF) kernel:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|_2^2) \quad (\gamma \text{ is a hyperparameter}) \tag{1.12}$$

In the Mahalanobis distance from Equation (1.11), $W$ is the parameter matrix to learn. It is of size $n \times n$ instead of $d \times d$ for the linear Mahalanobis metric learning approaches. So the number of parameters to learn is $O(n^2)$. This is nice for high-dimensional data but it might become impractical for applications with a very large number of points. Like $M$, $W$ must be PSD ($W \succeq 0$) in order to have a pseudo-distance.

From Equation (1.10), the nonlinear function $g$ can be expressed in terms of the matrix $N$ and the kernel matrix $K$ [19]:

$$g(x) = L(\phi(x)) = N\Phi^T(\phi(x)) = N(k(x_i, x))_{i=1}^n = NK_x \tag{1.13}$$

As mentioned, this chapter presents an overview of linear and nonlinear metric learning. The following chapter presents some state-of-the-art techniques for metric learning with multi-modal data.

# Chapter 2

# Metric learning for multi-modal data, state-of-the-art

Data are qualified as multi-modal when they arise from different heterogeneous sources (for example, songs represented by their audio features and their lyrics) or when they arise from the same source but are represented in different ways (for instance, a document represented in a bag-of-words vector or as a tf-idf vector). Simply concatenating the modalities to form a vector space and learning a uni-modal metric on this space might not give the best results, especially if the distance measure used is not the same in each modality. For example, the Euclidean distance can be the best suited distance measure for one modality while for another modality the cosine similarity is best-suited. A few metric learning techniques have been applied to multi-modal data. This chapter presents some state-of-the-art techniques for multi-modal metric learning.

## 2.1 Multiple Kernel Partial Order Embedding (MKPOE)

In [19], McFee and Lanckriet propose a multiple kernel approach, by embedding each modality of the data in separate feature spaces and combining all the feature spaces into a single optimized embedding space.

The embedding of each modality is done by using nonlinear feature maps $\phi^p$, $\forall p \in \{1, ..., m\}$ where $m$ is the number of different modalities. In order to have computationally cheaper operations by using the kernel trick, each feature map must satisfy a kernel function $k^p(x_i, x_j) = \langle \phi^p(x_i), \phi^p(x_j) \rangle$.

The way to combine the feature spaces is discussed later. But to explain the method in a simple way, the case where there is only one modality is considered first.

As in the kernelized metric learning approach, the feature map is followed by a linear transformation $L$ to map the data to $\mathbb{R}^{D'}$ and the parameterization $L = N\Phi^T$ is used to optimize the matrix $N$ instead of $L$. The embedding function $g : \mathbb{R}^d \to \mathbb{R}^{D'}$ has the form:

$$g(x) = L(\phi(x)) = NK_x \tag{2.1}$$

Based on a set $\mathcal{R}$ of relative (i.e. triplet) constraints $(x_i, x_j, x_k)$, the goal is to find an embedding function such that the distance constraints are satisfied with a margin of 1:

$$\|g(x_i) - g(x_j)\|_2^2 + 1 \leq \|g(x_i) - g(x_k)\|_2^2 \quad \forall (x_i, x_j, x_k) \in \mathcal{R} \tag{2.2}$$

The squared Euclidean distance between points in the embedding space can be expressed in

terms of $N$ and the kernel matrix $K$:

$$\begin{aligned}\|g(x_i) - g(x_j)\|_2^2 &= \|NK_i - NK_j\|_2^2 \\ &= (K_i - K_j)^T N^T N (K_i - K_j)\end{aligned} \tag{2.3}$$

The distance constraint in (2.2) can be rewritten as:

$$(K_i - K_j)^T N^T N (K_i - K_j) + 1 \leq (K_i - K_k)^T N^T N (K_i - K_k) \tag{2.4}$$

Because it may be impossible to satisfy all the constraints in $\mathcal{R}$, a slack variable $\xi_{ijk} > 0$ is introduced for each constraint. The goal is to minimize the values of these slack variables. Additionally, a regularization term is included to avoid overfitting. The regularization term proposed is the squared Hilbert-Schmidt (HS) norm of $L$, equivalent to the Frobenius norm if $L$ is not an operator on an infinite-dimensional space:

$$\|L\|_{HS}^2 = tr(L^T L) = tr(\Phi N^T N \Phi^T) = tr(N^T N \Phi^T \Phi) = tr(N^T N K) \tag{2.5}$$

Since $N$ always appears in the form of inner products ($N^T N$) in the distance constraints (2.4) and the regularization term (2.5), a variable change $W = N^T N \succeq 0$ may be used to obtain a convex optimization problem expressed in terms of a positive semi-definite matrix $W \in \mathbb{R}^{n \times n}$, where n is the number of training examples. What gives the following optimization problem, named Kernel Partial Order Embedding (KPOE):

$$\begin{aligned}\min_{W,\xi} \quad & tr(WK) + \frac{\lambda}{|\mathcal{R}|} \sum_{(x_i,x_j,x_k)\in\mathcal{R}} \xi_{ijk} \\ \text{s.t.} \quad & d^2(\phi_i, \phi_j) \doteq (K_i - K_j)^T W (K_i - K_j), \\ & d^2(\phi_i, \phi_j) + 1 \leq d^2(\phi_i, \phi_k) + \xi_{ijk}, \\ & \xi_{ijk} \geq 0 \qquad\qquad \forall (x_i, x_j, x_k) \in \mathcal{R}, \\ & W \succeq 0\end{aligned} \tag{2.6}$$

$\lambda > 0$ is a trade-off parameter to control the balance between regularization and loss minimization. The output of this method is $W$. KPOE can be seen as a (uni-modal) nonlinear adaptation of the optimization problem (1.6) proposed by Schultz and Joachims in [21], except that the regularization is on $L$ instead of $M$.

With the parameter matrix $W$, it is possible to obtain $N$ by using the eigendecomposition $W = V \Lambda V^T$ where the columns of $V$ are the eigenvectors of $W$ and $\Lambda$ is a diagonal matrix whose diagonal entries are the eigenvalues of $W$. Because $W = N^T N = V \Lambda V^T$, this leads to the following embedding function:

$$g(x) = NK_x = \Lambda^{1/2} V^T K_x \tag{2.7}$$

Looking now at the multi-modal case, where each modality has its own feature map $\phi^p(x)$ and the associated kernel matrix $K^p$, several techniques can be used to combine the feature spaces together. Here are the different possibilities proposed by the authors.

**Unweighted combination**

In the first possibility, the feature spaces are combined by a simple concatenation:

$$\phi(x_i) = (\phi^1(x_i), \phi^2(x_i), \ldots, \phi^m(x_i)) \tag{2.8}$$

This combination results in the *sum-kernel*, an unweighted sum of the base kernels:

$$\hat{K} = \sum_{p=1}^{m} K^p \tag{2.9}$$

because the inner product between two points in the embedding space is given by:

$$\langle \phi(x_i), \phi(x_j) \rangle = \sum_{p=1}^{m} \langle \phi^p(x_i), \phi^p(x_j) \rangle \tag{2.10}$$

A single projection $L$ is learned on this concatenation of feature spaces and the kernel $\hat{K}$ can be used in the optimization problem KPOE (2.6). The embedding function can be written as:

$$g(x) = L((\phi^p(x))_{p=1}^m) \tag{2.11}$$

**Weighted combination**

The previous technique considers the different kernels equally. A different combination is to consider a positive-weighted sum of the base kernels:

$$\hat{K} = \sum_{p=1}^{m} \mu_p K^p \ (\mu_p \geq 0) \tag{2.12}$$

Because some modalities can be more important than others, their contribution to the kernel is weighted in order to increase the quality of the resulting kernel. The optimization is done simultaneously on $W$ and on the weights $\mu_p$.

Figure 2.1 illustrates this technique. If the weights are set to $\mu_p = 1 \ \forall p \in \{1, \ldots, m\}$, then the figure represents the unweighted combination.

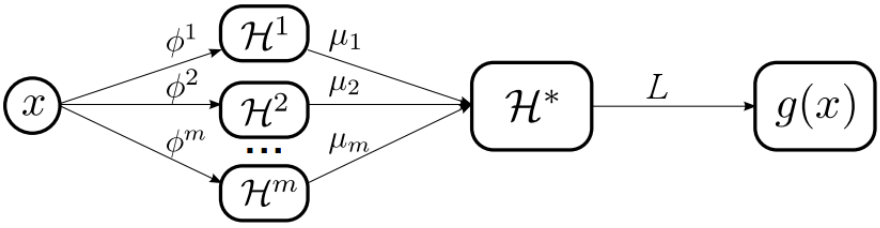



Figure 2.1: Weighted combination (image adapted from [19])

As $\hat{K}$ can be seen as:

$$\begin{aligned} \hat{K} &= \sum_{p=1}^{m} \mu_p \langle \phi(x_i), \phi(x_j) \rangle \\ &= \sum_{p=1}^{m} \langle \sqrt{\mu_p}\phi(x_i), \sqrt{\mu_p}\phi(x_j) \rangle \end{aligned} \tag{2.13}$$

the embedding function is:

$$g(x) = L((\sqrt{\mu_p}\phi^p(x))_{p=1}^m) \tag{2.14}$$

But the distance constraints in KPOE (2.6) contain differences of terms cubic in the optimization variables $W$ and $\mu_p$ when using this kernel. The problem is then non-convex and difficult to optimize.

**Concatenated projections**

Another possibility is to learn one projection matrix $L^p$ per modality $p$ and to concatenate the projections $L^p(\phi^p(x))$ rather than projecting the concatenation of $\phi^p$. This is illustrated in Figure 2.2. The learned projections $L^p$ are jointly optimized to produce the embedding space. The embedding function can be expressed as a concatenation of the learned projections:

$$g(x) = (L^p(\phi^p(x)))_{p=1}^m \tag{2.15}$$

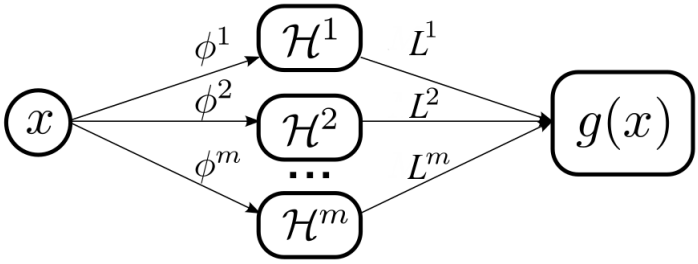



Figure 2.2: Concatenated projection (image adapted from [19])

Adapting the KPOE optimization problem (2.6) to multiple modalities leads to the Multiple Kernel Partial Order Embedding (MKPOE) optimization problem:

$$\begin{aligned}
\min_{W^p,\xi} \quad & \sum_{p=1}^{m} tr(W^p K^p) + \frac{\lambda}{|\mathcal{R}|} \sum_{(x_i,x_j,x_k)\in\mathcal{R}} \xi_{ijk} \\
\text{s.t.} \quad & d^2(\phi_i,\phi_j) \doteq \sum_{p=1}^{m} (K_i^p - K_j^p)^T W^p (K_i^p - K_j^p), \\
& d^2(\phi_i,\phi_j) + 1 \leq d^2(\phi_i,\phi_k) + \xi_{ijk}, \\
& \xi_{ijk} \geq 0 \qquad\qquad \forall (x_i,x_j,x_k) \in \mathcal{R}, \\
& W^p \succeq 0 \qquad\qquad \forall p \in \{1,\ldots,m\}
\end{aligned} \tag{2.16}$$

This is the method chosen by the authors for their multiple-kernel formulation since it outperforms the unweighted combination rule and it is easier to optimize than the weighted combination rule.

The problem (2.16) is optimized thanks to a subgradient descent on $W^p$, a generalization of the gradient descent to non-differentiable functions, followed by a projection onto the feasible set $W^p \succeq 0$.

### 2.1.1 Diagonal matrices to reduce computational cost

MKPOE needs to optimize $m$ different matrices $W^p \in \mathbb{R}^{n\times n}$, which can become problematic for large datasets (large $n$) in term of computational complexity. Indeed, constraining the matrices $W^p$ to be positive semi-definite requires to project them onto the feasible set $W^p \succeq 0$ at each iteration. This projection has a total time complexity of $O(mn^3)$ because the projection of one matrix $W^p$ is done by computing its eigendecomposition, which has a time complexity of $O(n^3)$, and setting all the negative eigenvalues of $W^p$ to 0.

In order to reduce the time complexity of the algorithm, the authors propose to learn only diagonal $W^p$. A diagonal matrix is PSD if its diagional values are non-negative. Replacing the constraints $W^p \succeq 0$ by linear constraints $W_{ii}^p \geq 0$ leads to a total time complexity of $O(mn)$ at each iteration for the projection onto the feasible set. This modification reduces the flexibility of

the model as it comes down to a simple weighting of the contribution of a training point in one modality to the metric, but it results in a more efficient optimization.

## 2.2 Multi-wing harmonium model

In MKPOE, multiple high-dimensional PSD matrices are optimized in order to learn a Mahalanobis distance in each feature space. Projecting each matrix $W^p$ onto the feasible set is computationally expensive. To cope with this cost, the solution proposed by Xie and Xing [28] to learn a multi-modal metric is based on a multi-wing harmonium model (or restricted Boltzmann machine).

A Boltzmann machine is a type of (generative) stochastic artificial neural network. It is stochastic because the units have a stochastic activation function, meaning that they have a particular distribution (for example binary or Gaussian). Restricted Boltzmann machines (RBM) have the restriction that their network forms a bipartite graph: the 2 groups of units are called the "visible" and the "hidden" units. The learning consists of an alternation between the propagation step, where hidden units are turned on given visible units, and the reconstruction step, where visible units are turned on given hidden units. The connection weights are adjusted to minimize the reconstruction error. The idea behind RBM is that there are two sets of variables (the visible and the hidden units) and it learns how the two sets of variables are connected to each other.

In the multi-wing harmonium model proposed in [28], the data features of each modality correspond to a portion of the visible units (one wing) and they are all connected to the same hidden units. The hidden units represent the shared latent space in which the data are represented when taking all the modalities into account. This is the latent space that needs to be learned by the algorithm in order to optimize the distance between points based on the distance constraints. Figure 2.3 illustrates the multi-wing harmonium model for 2 modalities of the data. The first modality is represented by a vector $u = (u_1, \ldots, u_{d'}) \in \mathbb{R}^{d'}$ and the second modality is represented by a vector $v = (v_1, \ldots, v_{d''}) \in \mathbb{R}^{d''}$. $g(x_i)$, the embedding of $x_i$ in the latent space, is given by the expectation of $(h_1, \ldots, h_{D'}) \in \mathbb{R}^{D'}$, where $D'$ is an hyperparameter that represents the dimension of the latent space.

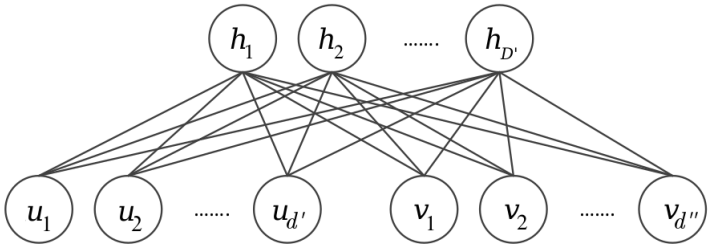



Figure 2.3: Dual-wing harmonium model (image adapted from [28])

The distance constraints provided are sets of similar ($\mathcal{S}$) or dissimilar ($\mathcal{D}$) pairs. The distance between similar points should be minimized in the latent space while dissimilar points should be separated by a margin of 1. They define the relaxed optimization problem as:

$$\begin{aligned} \min_{\Theta} \quad & \frac{1}{|\mathcal{X}|}\ell(\mathcal{X}, \Theta) + \lambda_1 \frac{1}{|\mathcal{S}|} \sum_{(x_i,x_j)\in\mathcal{S}} \|g(x_i) - g(x_j)\|_2^2 + \lambda_2 \frac{1}{|\mathcal{D}|} \sum_{(x_i,x_j)\in\mathcal{D}} \xi_{ij} \\ \text{s.t.} \quad & \|g(x_i) - g(x_j)\|_2^2 \geq 1 - \xi_{ij}, \\ & \xi_{ij} \geq 0 \qquad \forall (x_i, x_j) \in \mathcal{D} \end{aligned} \tag{2.17}$$

In problem (2.17), the loss function $\ell(\mathcal{X}, \Theta)$ is the negative log-likelihood of the training data $\mathcal{X}$ given the parameters $\Theta$. It is used as regularizer to avoid overfitting by having a model that explains well the initial data. The hyperparameters $\lambda_1$ and $\lambda_2$ control the trade-off between the regularization and the satisfaction of the similarity or dissimilarity distance constraints. $g$ is a function of the set of parameters $\Theta$, learned when optimizing the problem. The optimization is also performed by using a subgradient method. More details on this technique can be found in the original paper [28].

Compared to MKPOE, the set of distance constraints used are $\mathcal{S}$ and $\mathcal{D}$, instead of a set of relative constraints $\mathcal{R}$. The main difference is that it directly learns a nonlinear transformation through the restricted Boltzmann machine, while MKPOE learns a linear transformation in the space induced by kernel functions. The learned metric is not a Mahalanobis distance. An advantage is that there is no need for projections onto the set of PSD matrices, which is a costly operation in metric learning algorithms.

The authors say that their solution is efficient and yields good results. But it is far from the usual metric learning approaches, such as learning a Mahalanobis distance function. Furthermore, some techniques have been proposed to improve the execution time of MKPOE, such as online methods (explored in chapter 5). For these reasons, the multi-wing Harmonium technique will not be further investigated and this master's thesis focuses on the first technique: MKPOE.

# Chapter 3

# Experimental setting

This chapter describes the experimental setting used to assess the performance of the metric learning techniques in term of accuracy and computation time. It begins with a description of the *Million Song Dataset* (section 3.1), the dataset used throughout this master's thesis, followed by a description of the classification tasks chosen to assess the performance of the algorithms (section 3.2). Section 3.3 contains a description of kNN, the classification algorithm used for these tasks. Then there is a detailed description of the datasets used for the tasks (section 3.4), a note on the standardization and normalization of the features (section 3.5) and a description of how the distance constraints are generated (section 3.6). The last section of this chapter explains how the step size of the iterative optimization algorithm is fixed.

## 3.1 Million Song Dataset

The *Million Song Dataset* (MSD) [5] is a collection of features of a million songs, provided by *The Echo Nest*[1]. This dataset fits well with the objective of this master's thesis because it consists of multi-modal data: each song is represented by audio features and heterogeneous metadata. The dataset doesn't contain the audio of the songs but only the derived features. This is convenient for the task as there is no need to take care of the extraction of the audio features. Examples of the available audio features are the timbre, pitch, energy, tempo, loudness and danceability. Examples of metadata features are the year of the song and its duration.

In addition, a number of complementary datasets are provided. The *musiXmatch* dataset[2] contains song lyrics in bag-of-words format: each song is represented by a vector containing the counts of the words from the 5000 most frequent words across the MSD. These lyrics can be matched with songs of the MSD. The *MSD Allmusic Top Genre Dataset (Top-MAGD)* [3] provides the genre associated to each song. Only a subset of the songs of the MSD are available in the Top-MAGD dataset because the genres were collected from `Allmusic.com` whose songs don't totally match with those of the MSD.

Since the whole dataset is more than 300GB, a 1% subset of the MSD is provided (10000 songs). This subset is used for the experiments.

[1] `http://the.echonest.com/`

[2] musiXmatch dataset, the official lyrics collection for the Million Song Dataset, available at: `http://labrosa.ee.columbia.edu/millionsong/musixmatch`

[3] `http://www.ifs.tuwien.ac.at/mir/msd/`

## 3.2 Tasks

In order to assess the performance of the metric learning techniques, they are tested on two classification tasks: artist recognition and genre recognition. This section presents the two tasks.

### 3.2.1 Artist recognition

The artist recognition task is a multi-class classification task proposed in the MSD where the goal is to predict the artist of the songs in the test set. The MSD is composed of songs from many artists, many of them having few songs. Therefore, this classification task is difficult, in the sense that it is hard to reach a high accuracy.

### 3.2.2 Genre recognition

Because there are far fewer genre labels than artists in the MSD, another task considered is genre recognition. The goal is to predict the genre of the songs in the test set. This task is proposed by the team behind the Top-MAGD dataset, which provides the genres corresponding to the songs of the MSD[4].

Table 3.1 contains the list of genres with the number of songs contained in the intersection between the 1% subset of the MSD and the Top-MAGD dataset. This table shows that the dataset is highly imbalanced, so most of the experiments are done on a randomly sampled balanced dataset.

| **Genre** | **Songs** | **Genre** | **Songs** |
|---|---|---|---|
| Pop_Rock | 1926 | RnB | 132 |
| Latin | 257 | Blues | 126 |
| Rap | 236 | Reggae | 120 |
| Electronic | 228 | New Age | 95 |
| Country | 175 | Folk | 48 |
| Jazz | 173 | Vocal | 36 |
| International | 147 | | |

Table 3.1: Summary of the number of songs of each genre in the 1% subset of the MSD

## 3.3 k-nearest neighbors

The classification algorithm used for the two tasks is the k-nearest neighbors algorithm [8]. It classifies a test example by a majority vote of the classes from its k nearest neighbors among the training examples, where k is a hyperparameter to tune. Without prior knowledge about the similarity between data points of the training set, the nearest neighbors are often chosen by using the Euclidean distance. This distance isn't necessarily the optimal distance to use, especially when different tasks use the same set of features but the distance between the data points shouldn't be the same for each task. So a metric learning preprocessing step is well adapted to this classification algorithm.

Also, kNN is convenient for multi-class classification because it supports more than 2 classes without taking more time to train.

[4]Available at: `http://www.ifs.tuwien.ac.at/mir/msd/download.html`

## 3.4 Datasets

For the first experiments, smaller datasets with well-balanced classes are created. For the artist recognition task, artists having at least 8 songs are selected, and for each artist, 8 songs are randomly selected. This results a dataset of 99 classes and 792 songs. For the genre recognition task, in order to have a binary classification task without too many examples, the songs from the classes "Pop_Rock" and "Rap" are selected because they are among the genres with the most songs. The number of songs in the two classes are balanced by limiting the number of songs in the "Pop_Rock" class. This results in a dataset of 472 songs (236 songs in each class).

For the features, the artist recognition baseline method uses audio features computed from the timbre vectors. Using the average and the covariance of the timbre vectors over all the segments of a song gives 90 audio features per song. On top of these audio features, other individual audio features or metadata features that could represent well a song to recognize its artist or genre are added: the year of the song, its danceability (a real value between 0 and 1), its energy (also a real value between 0 and 1) and its tempo. The lyrics are also added to the features in order to fully take advantage of the multi-modal aspect of the dataset. The lyrics are represented as a smoothed tf-idf vector. For each song $x_i$, each term $w$ from the 5000 most frequent words has a tf-idf value computed by following the formula from the *scikit-learn* documentation [5]

$$\text{tf-idf}(x_i,\, w) = \text{tf}(w) \cdot \left( \log \left( \frac{n+1}{\mid \{x_i \in \mathcal{X} : w \in x_i\} \mid +1} \right) + 1 \right) \tag{3.1}$$

where $\text{tf}(w)$ is the term count of the term $w$ in the lyrics of song $x_i$ and $n$ is the number of songs in the training set $\mathcal{X}$. This tf-idf is smooth thanks to the 1 added to the numerator and the denominator to avoid having divisions by 0 when a term is present in none of the songs of the training set.

Many songs do not have lyrics. This can happen for two reasons: copyright problems or instrumental songs, but it is impossible to differentiate the two. The tf-idf vectors of these songs are set to zeros. For instrumental songs this makes sense because it means that all the words are not present in the song and have thus a count of 0.

## 3.5 Standardization and normalization

All the features except the lyrics are standardized by removing the mean value of each feature and then by dividing by its standard deviation. All the standardized features have a mean value of 0 and unit variance.

The tf-idf vectors of the lyrics are normalized using the l2-norm because their similarity is measured by a cosine similarity where the length of the lyrics should not be taken into account.

**Definition 2.** The *cosine similarity* between two vectors $x_i$ and $x_j$ is defined as

$$S(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2} \tag{3.2}$$

## 3.6 Distance constraints

MKPOE works with triplet distance constraints. Those constraints are generated to fit with the classification tasks. Therefore, triplet constraints $(x_i, x_j, x_k)$ are generated, where $x_i$, $x_j$ and

[5] `http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfTransformer.html`

$x_k$ are data points from the training set, such that data points $x_i$ and $x_j$ are in the same class ($y_i = y_j$) and data point $x_k$ is in a different class than $x_i$ ($y_i \neq y_k$).

The datasets are splitted into a training set and a test set with a stratified 3-fold cross-validation procedure. The folds of the split (except the last one) result in a training set size of 495 examples for the artist recognition task (5 examples per class) and 314 examples for the genre recognition task (157 examples per class). The number of triplet constraints generated from the training set is 970,200 for the first task and 7,690,488 for the second task.

The number of triplet constraints is large. This leads to a higher execution time for MKPOE which has a time complexity proportional to the number of constraints. The approach from LMNN [24] is adopted to reduce the number of constraints: for each data point $x_i$, $k$ target neighbors are selected, which are the $k$ nearest neighbors from the data point $x_i$ that have the same class label than $x_i$. These targets neighbors are the data points that should be near $x_i$ in the embedding space. The value of $k$ can be fixed. All the data points from another class than $x_i$ are called the impostors and should be far from $x_i$ in the embedding space. Creating triplet constraints $(x_i, x_j, x_k)$ by selecting 3 targets neighbors for each data point $x_i$ and all the points from another class as impostors results in a total of 727,650 constraints for the first dataset and 147,894 constraints for the second dataset. With this approach, the number of constraints for the genre recognition task is decreased much more than the number of constraints for the artist recognition task. This is simply because the training set of the artist recognition task contains 5 examples per class, so selecting 3 target neighbors instead of 4 doesn't change the number of constraints much, contrary to the training set of the other task which contains 157 examples per class.

Another popular choice (used in [22] and [23]) to reduce the number of constraints is to limit the number of impostors in addition to the number of target neighbors, by considering only a small number of closest points from a different class. This number of impostors is not necessarily the same as the number of targets neighbors. By selecting 3 impostors for each data point $x_i$, the number of triplet constraints is further decreased to 4455 constraints for the first dataset and 2826 constraints for the second dataset. It should be noted that this approach isn't optimal because a data point from a different class than $x_i$ that isn't selected as impostor can become closer to $x_i$ than its target neighbors in the embedding space. However, this gives nice results, as shown in the next chapter.

The table 3.2 contains a summary of the two datasets. all_constraints, all_impostors and 3_impostors denote the number of constraints in each of the 3 situations from above.

| | Artist recognition | Genre recognition |
|---|---|---|
| training examples | 495 | 314 |
| test examples | 297 | 158 |
| classes | 99 | 2 |
| features | 5094 | 5094 |
| all_constraints | 970200 | 7690488 |
| all_impostors | 727650 | 147894 |
| 3_impostors | 4455 | 2826 |

Table 3.2: Summary of the two datasets obtained with the first fold of a stratified 3-fold split

## 3.7 Convergence of the subgradient descent

The authors of MKPOE [19] propose to use a projected subgradient descent to optimize the convex problem (2.16). At each iteration, a subgradient descent step is performed on the objective function, followed by a projection onto the set of PSD matrices. As stated earlier, the projection step is done by computing the eigendecomposition of $W^p$ and setting all the negatives eigenvalues to 0.

A choice that needs to be made is how to fix the step size (or learning rate) in order for the subgradient descent to converge. This section presents several possibilites to fix the step size and the one chosen in the implementation of MKPOE.

One possibility is to use a constant step size. The problem with this approach is the difficulty to choose the right value. If the step size is too small the convergence will be slow. The iteration algorithm can even be inefficient if it doesn't go down enough (meaning that the value of the objective function doesn't decrease enough) after a fixed maximum number of iterations. On the other side, if the step size is too large, the optimization can diverge from the minimum of the objective function. Furthermore, the scale of the step size changes with the values of the hyperparameters, such as the regularization trade-off hyperparameter C, because they change the scale of the gradient. So in a cross-validation scheme it is not easy to fix a good value for the step size that depends of the hyperparameter.

Another possibility is to use a decreasing step size. This way if the initial value is too large it will improve after some iterations. Also, the steps near the minimum of the objective function, i.e. the steps at the end of the iterations, will be smaller than at the beginning of the iterations, which can help to avoid divergence. For example, a usual method to choose the step size at iteration $t$ is

$$\alpha_t = \frac{c}{t+1} \quad \text{s.t. } c > 0 \tag{3.3}$$

with $c$ a constant. But the step size often decreases too fast with this method, which leads to slow convergence. To make it decrease more slowly, a geometric decreasing sequence can be used:

$$\alpha_t = c^{t+1} \quad \text{s.t. } 0 < c < 1 \tag{3.4}$$

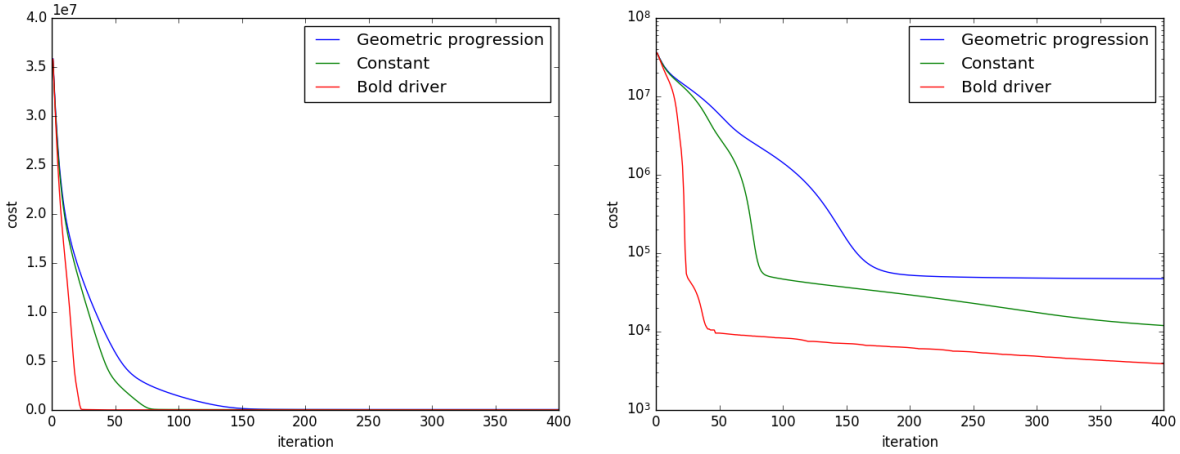
Good values for $c$ are 0.95 or 0.99. But if it decreases too quickly, the steps will be too small and the gradient descent won't go down enough during the optimization. If the decrease is too slow, the step size might become too large, which may result in a divergence of the iteration algorithm (the value of the objective function starts to oscillate).

The optimal step size at each iteration can be found by line search. This is done by finding the step size that gives the minimal value of the objective function in the direction of the search, by using an algorithm such as a binary search. The problem with this approach is that it is too computationally expensive.

The proposed solution to avoid these problems is to use an adaptive learning rate. If after an iteration the cost function decreases, the step size $\alpha$ is multiplied by a factor 1.1. On the other hand, each time the cost function increases after an iteration, it means that the step size is too large, and $\alpha$ is decreased by half and the iteration cancelled. The justification behind why it is decreased by half is because if the objective function is symmetric around the minimum, then dividing the step size by 2 will probably bring $W_p$ (the matrix to optimize) near the minimum. This way, the value of the learning rate is easily fixed and adapts automatically to the situation. This technique, kwown as the "bold driver" method, is proposed in [1]. Note that this technique

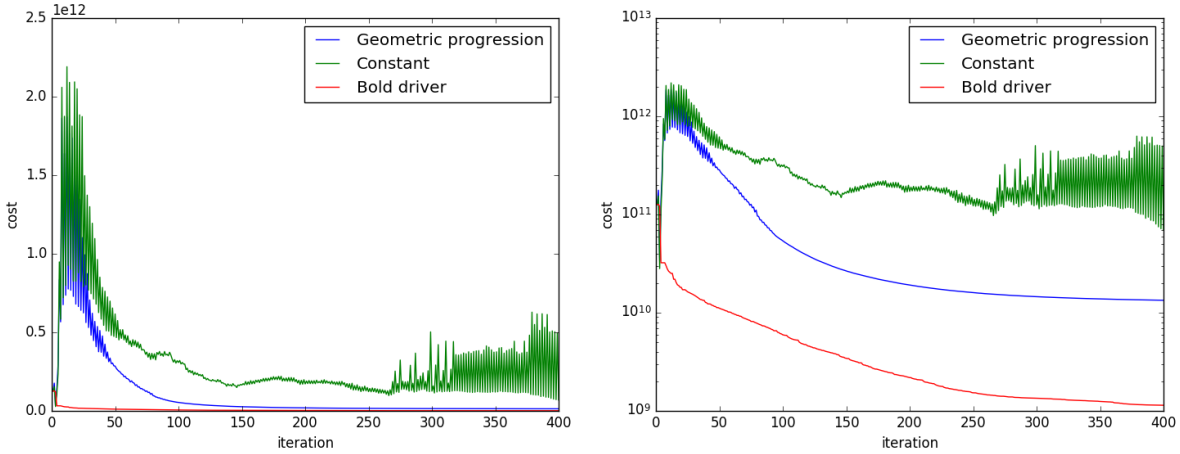
is also the one used in the solver of LMNN [25].

Figure 3.1 shows the convergence of the projected subgradient descent obtained with the 3 methods on the dataset of the genre recognition task using the all_impostors set of constraints. For the constant step size, $c$ is fixed to 1 and for the geometric progression it is fixed to 0.99. The geometric progression gives poor performance in this case because the learning rate decreases too quickly. The performance of a constant step size are not as good as with the "bold driver" method.




(a) Normal scale (b) Log scale

Figure 3.1: Convergence of the subgradient descent on the genre recognition dataset with the all_impostors set of constraints




(a) Normal scale (b) Log scale

Figure 3.2: Convergence of the subgradient descent on the genre recognition dataset with the 3_impostors set of constraints

Figure 3.2 shows the convergence using the 3_impostors set of constraints. $c$ is fixed to the same values as before. Again the constant step size and the geometric progression have low performance compared to the "bold driver" method. In this case, the step size obtained with the geometric progression is too small at the end, which leads to poor convergence, and with the constant step size the iteration algorithm diverges, as it can be observed by the oscillations in

the value of objective function after some iterations.

The "bold driver" method works well compared to the other methods tested. The main advantage is that there is no need to worry about the value given to the constant step size or the value of $c$ for the decreasing step size. This is especially useful when the dataset or the hyperparameters change (for example during a grid search), because the step size is sensitive to these changes.

Other approaches might be explored to find the learning rate. Some future work could focus on the exploration of better optimization techniques than a simple subgradient descent, but this is not covered in this master's thesis.

# Chapter 4

# How to combine the modalities?

The datasets used have features coming from multiple modalities of the data: audio features, semantic features such as lyrics,... This chapter studies if the multi-modal aspect of the data can be used to increase the accuracy of the two classification tasks at hand and looks at the best way to combine the modalities.

The first section of this chapter presents the case where the multi-modal aspect of the data is ignored and the features are all concatenated together to form a vector space. In the second section, the multi-modal aspect of the data is taken into account and different possible formulations to combine the modalities are presented. The results obtained on the two datasets with each formulation are reported in the third section, while section 4.4 contains an in depth analysis of the results obtained with MKPOE. Section 4.5 contains the implementation details, with particular attention to how the cross-validation is performed. Finally, the last section concludes this chapters by summarizing the main results obtained.

## 4.1 One kernel for all the features

If the modalities of the data are not taken into account, all the features from all the modalities are concatenated to form one vector space representing the data. The KPOE (2.6) metric learning technique can be used with the condition of choosing a kernel that represents the nonlinear transformation of the input data. Two kernels are tested in the experiments:

- **Linear kernel**: $K(x_i, x_j) = \langle x_i, x_j \rangle$.
- **RBF kernel**: $K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|_2^2)$ where $\gamma$ is a hyperparameter to tune.

The output of KPOE is the learned parameter matrix $W$ of size $n \times n$ (where $n$ is the size of the training set).

## 4.2 Multiple formulations to combine the modalities

The data come from heterogeneous sources. Integrating these modalities together into one feature space may not give the best results when using KPOE, for the two following reasons:

- With only one kernel, the similarity between objects can be altered by irrelevant features. This is especially true when no feature selection is performed.
- When using one kernel to represent the similarity between objects it is not clear which kernel should be used. Also, the similarity between objects can be expressed differently on each modality. For example the similarity between songs is not the same on the audio features

than on the semantic features such as lyrics. Instead, by having some prior knowledge about each modality, different kernels can be choosen for each modality.

The objective here is to see if considering the modalities separately can increase the accuracy and how to combine the modalities.

Each modality $p$ has one kernel $K^p$ associated to it, based on prior knowledge of the data contained in each modality. Here are the 3 modalities used for the set of features described in the previous chapter and their kernels:

- **Audio**: this modality contains the average and covariance of the timbre vectors. It has 90 features. The selected kernel function between songs in this modality is the RBF kernel.

- **Semantic**: this modality contains the lyrics of the songs as normalized tf-idf vectors. It has 5000 features. The songs can be compared in this modality by using the cosine similarity. Because the tf-idf vectors are normalized, i.e. $\|x_i\|_2 = \|x_j\|_2 = 1$, the cosine similarity is simply the dot product between $x_i$ and $x_j$:

$$S(x_i, x_j) = \frac{x_i^T x_j}{\|x_i\|_2 \|x_j\|_2} = x_i^T x_j \tag{4.1}$$

  The selected kernel between songs in this modality is the linear kernel.

- **Other**: this modality contains individual audio features and metadata features (such as the year of the song). The features chosen in the experiments are the tempo, the year, the danceability and the energy of songs. The selected kernel between songs in this modality is again a RBF kernel.

For each RBF kernel, the set in which the hyperparameter $\gamma$ is chosen by cross-validation is $\gamma \in \{10^{-3}, 10^{-2}, ..., 10\}$.

It is not really clear what the best method is to combine those modalities. Therefore the results of two formulations proposed in [19] are compared: KPOE with an unweighted sum of kernels and MKPOE.

### 4.2.1 Unweighted sum of kernels

When concatenating the feature maps for each modality $p \in 1, ..., m$:

$$\phi(x_i) = (\phi^p(x_i))_{p=1}^m \tag{4.2}$$

the kernel is computed as:

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle = \sum_{p=1}^{m} \langle \phi^p(x_i), \phi^p(x_j) \rangle = \sum_{p=1}^{m} k^p(x_i, x_j) \tag{4.3}$$

The resulting unweighted sum kernel is:

$$\hat{K} = \sum_{p=1}^{m} K^p \tag{4.4}$$

This kernel $\hat{K}$ can be used in the KPOE (2.6) metric learning technique. With this approach, only one parameter matrix $W$ of size $n \times n$ has to be learned.

### 4.2.2 MKPOE

In the formulation of MKPOE (2.16), the feature maps of each modality are not concatenated before learning a linear projection $L$, but a different $L^p$ is learned for each modality $p$ and the projections $L^p(\phi^p(x))$ are concatenated. Compared to the unweighted sum of kernels, this technique is more flexible as it learns multiple projections. It should give better results, especially if some modalities are irrelevant for the task. Indeed, the projection learned can give little weight to an irrelevant modality, reducing its impact on the learn metric, while in the unweighted sum of kernels the unique projection cannot choose the weights of each modality. This difference also applies to the initial case where a unique kernel is chosen on the concatenated features.

Separating the features into more modalities should give better results because the model would be more flexible. For example, in the third modality *Other*, the year may be irrelevant to predict the genre of a song, while the tempo is important, it is then better to separate them into different modalities. Since the hyperparameters of each kernel must be tuned and a matrix $W^p$ of size $n \times n$ must be learned per modality, the number of modalities is limited to 3 in the experiments.

## 4.3 Results

Tables 4.1 and 4.2 contain the experimental results obtained with each approach of the previous section as well as the results obtained by a kNN classifier without a metric learning step.

| Method | Constraints | accuracy | SD | time (1 run) | total time |
|---|---|---|---|---|---|
| No metric learning | - | **0.250** | 0.009 | 10ms | 3s |
| KPOE linear kernel | all_impostors | 0.238 | 0.029 | 18min42 | 7h47 |
| | 3_impostors | 0.163 | 0.012 | 39s | 0h22 |
| KPOE RBF kernel | all_impostors | 0.190 | 0.019 | 21min31 | 13h07 |
| | 3_impostors | 0.195 | 0.017 | 43s | 0h37 |
| KPOE $\Sigma K$ | all_impostors | 0.213 | 0.025 | 19min18 | 36h30 |
| | 3_impostors | 0.236 | 0.008 | 41s | 1h41 |
| MKPOE | all_impostors | 0.178 | 0.007 | 35min09 | 54h55 |
| | 3_impostors | 0.228 | 0.012 | 1min41 | 4h11 |

Table 4.1: Artist recognition: experimental results of metric learning with different combinations of the modalities (SD = standard deviation)

| Method | Constraints | accuracy | SD | time (1 run) | total time |
|---|---|---|---|---|---|
| No metric learning | - | 0.865 | 0.029 | 8ms | 1.5s |
| KPOE linear kernel | all_impostors | 0.879 | 0.031 | 2min47 | 1h14 |
| | 3_impostors | 0.888 | 0.034 | 17s | 0h09 |
| KPOE RBF kernel | all_impostors | 0.885 | 0.02 | 2min57 | 1h59 |
| | 3_impostors | 0.894 | 0.028 | 19s | 0h14 |
| KPOE $\Sigma K$ | all_impostors | 0.900 | 0.021 | 2min54 | 6h17 |
| | 3_impostors | 0.890 | 0.031 | 17s | 0h42 |
| MKPOE | all_impostors | **0.905** | 0.008 | 4min20 | 8h38 |
| | 3_impostors | 0.894 | 0.016 | 43s | 2h14 |

Table 4.2: Genre recognition: experimental results of metric learning with different combinations of the modalities (SD = standard deviation)

In the tables, KPOE $\Sigma K$ corresponds to KPOE using the unweighted sum of kernels, $\hat{K}$ from

Equation (4.4). The first line contains the results without metric learning, the second and third lines correspond to the two kernels tested on a uni-modal approach using KPOE and the two last lines correspond to multi-modal approaches. The experimental tests are not performed on the full set of constraints but limited to the sets all_impostors and 3_impostors instead because of the time taken by the algorithm, but the results are expected to be close to those obtained with all_impostors.

In all cases, the trade-off hyperparameter $\lambda$ needs to be tuned as well as the $k$ of the kNN algorithm. The set of values tested for $\lambda$ is $\{1, 10^3, 10^4, \ldots, 10^7\}$ and $k$ is tested in the range $\{1, 2, \ldots, 10\}$. The accuracy displayed for each algorithm is the mean accuracy of a stratified 3-fold cross-validation where in each fold, the hyperparameters are tuned by using another stratified 3-fold cross-validation. Next to the accuracy is displayed the standard deviation (SD) of the accuracies obtained in each fold. The execution time of each algorithm on one fold of the split is reported in the column "time (1 run)". The column "total time" gives the total execution time of the nested 3-fold cross-validation. The total time depends on the number of hyperparameters of each algorithm and on the number of values tested for each of these hyperparameters. The implementation details are explained in the last section (4.5) of this chapter. The experiments are performed on a machine with a 3.4Ghz Intel Core i7 processor and 16GB of RAM.

For the artist recognition, Table 4.1 shows that all the tested combinations give an accuracy on the test set worse than the one obtained without a metric learning step. As shown in the next section, this is because the metric learning overfits the training set and is not able to generalize to the test set. This overfitting is due to the small number of examples per class compared to the high-dimensionality of the space. MKPOE does not improve the accuracy on the artist recognition task for this dataset.

As shown in Table 4.2, the results on the genre recognition are better. Indeed, in the case of the genre recognition, all the methods improve the accuracy when compared without a metric learning step. The accuracy obtained with all the impostors as constraints is not really higher than with 3 impostors and it is even worse when the features are concatenated (i.e. KPOE with a linear or a RBF kernel). But the difference in execution time between those 2 sets of constraints is high. In practice the number of constraints can be restricted without impacting too much the accuracy of the classifier.

From all the tested combinations, MKPOE gives the best results on the genre recognition task: +4% in accuracy. However, the accuracy gained with MKPOE is not much larger than the accuracy gained with the other methods but MKPOE takes much more time.

### 4.3.1 Statistical comparisons

A statistical test can be applied to see if some of the proposed algorithms are significantly better than the others. Following [11], it is recommended to use the Friedman test when comparing multiple algorithms over multiple datasets. For each dataset (or task in this case), a rank is assigned to each algorithm, such that the rank 1 is assigned to the algorithm with the highest accuracy, the rank 2 is assigned to the second best algorithm, etc. In case of ties in the ranks, average ranks are assigned. Let $r_i^j$ be the rank of algorithm $j$ (out of $k$ algorithms) on the dataset

$i$ (out of $N$). In this case $k = 9$ and $N = 2$. The improved Friedman statistic $F_F$ is given by:

$$\chi_F^2 = \frac{12N}{k(k+1)} \left( \frac{1}{N^2} \sum_{j=1}^{k} \left( \sum_{i=1}^{N} r_i^j \right)^2 - \frac{k(k+1)^2}{4} \right) \tag{4.5}$$

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2} \tag{4.6}$$

$F_F$ is distributed according to the F-distribution with $(k-1)$ and $(k-1)(N-1)$ degrees of freedom. But since there are ties in the ranks, it is proposed in [7] to use the corrected formula (4.7) instead of the $\chi_F^2$ statistic of formula (4.5).

$$\chi_F^2 = \frac{N(k-1)}{(\sum_{i=1}^{N} \sum_{j=1}^{k} r_{ij}^2 - C_F)} \left( \frac{1}{N} \sum_{j=1}^{k} \left( \sum_{i=1}^{N} r_i^j \right)^2 - C_F \right) \tag{4.7}$$

where

$$C_F = \frac{Nk(k+1)^2}{4} \tag{4.8}$$

The null hypothesis states that all algorithms are equivalent (i.e. the measured average ranks are not significantly different from the mean rank), whereas the alternative hypothesis states that at least one algorithm is different from at least one other algorithm.

The test statistic has a value $F_F = 0.35$. It is distributed according to the F-distribution with $k - 1 = 8$ and $(k - 1)(N - 1) = 8$ degrees of freedom. By setting the significance level (i.e. the probability of rejecting the null hypothesis given that it is true) to $\alpha = 0.05$, the critical value of $F(8, 8)$ is 3.44. The null hypothesis can not be rejected because $F_F = 0.35 < 3.44 = F(8, 8)$. There is no statistical evidence that there is a difference between the algorithms.

In all cases, the metric learning algorithms learn a poor metric for the artist recognition task. Because of this, it is not expected that the metric learning algorithms are, in all cases, significantly better than kNN without a metric learning step. But for the genre recognition task, the results show that the metric learning step allows to have a better classifier. A statistical test is applied on the accuracies obtained on each fold for the genre recognition task to see if the difference between the classifiers is statistically different or not for this task.

Since there are 9 algorithms and 3 folds, a Friedman test can be applied with $k = 9$ and $N = 3$. Thus resulting in a F-distribution with $k - 1 = 8$ and $(k - 1)(N - 1) = 16$ degrees of freedom. This time, the test statistic is $F_F = 1.52$ and the critical value of $F(8, 16)$ under the signifance level $\alpha = 0.05$ is 2.59. Again, the null hypothesis cannot be rejected (because $1.52 < 2.59$), so the difference between the algorithms is not statistically significant, even if the obtained accuracies are higher with a metric learning step.

The number of tested datasets is small, so these approaches should be tested on other datasets to confirm if the difference is statistically significant or not.

## 4.4 Analysis of the results obtained with MKPOE

This section analyzes more in depth the results obtained with MKPOE. The training/test split used is the same as the first fold of the split from the previous section. The analysis is first done on the dataset of the genre recognition using the all_impostors set of distance constraints, then the results of MKPOE are analyzed on the training set of the artist recognition task.

### 4.4.1 Genre recognition

When the trade-off hyperparameter $\lambda$ is high enough and enough iterations are done during the sugradient descent, the number of satisfied distance constraints on the training set increases in the embedding space compared to the original space. This is illustrated in Figure 4.1. In total the metric learning algorithm allows to increase by 1106 the number of satisfied constraints. A lot of constraints are already satisfied in the original space (98.8%), which explains the accuracy obtained by the kNN algorithm without a metric learning preprocessing step (see Table 4.2).

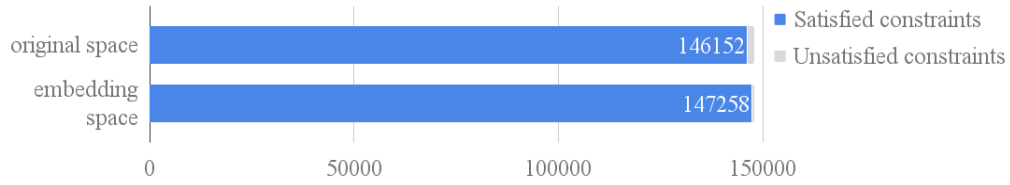
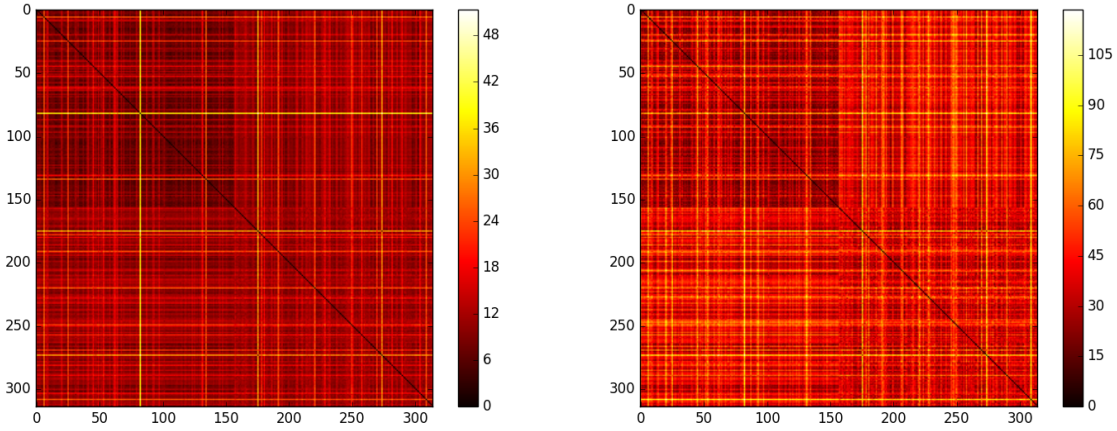



Figure 4.1: Genre recognition: number of satisfied constraints (from all_impostors) in the embedding space compared to the original space. Total number of constraints: 147894

The cross-validation performed on the first fold of the split in the previous section gives $k = 7$ as the best number of neighbors to use for the kNN. For each training point $x_i$, the number of points from the same class than $x_i$ that are among the 7 nearest neighbors of $x_i$ are analyzed before and after the metric learning phase. This analysis shows that, in general, the number of points in the 7-neighbordhood of $x_i$ that are in the same class than $x_i$ is bigger in the embedding space than in the original space. Indeed, the test shows that this number has increased for 113 training points out of 314, while it has decreased for only 25 training points and stays the same for 176 points. MKPOE performs well to group points of the same class near each other while pushing apart points from different classes.

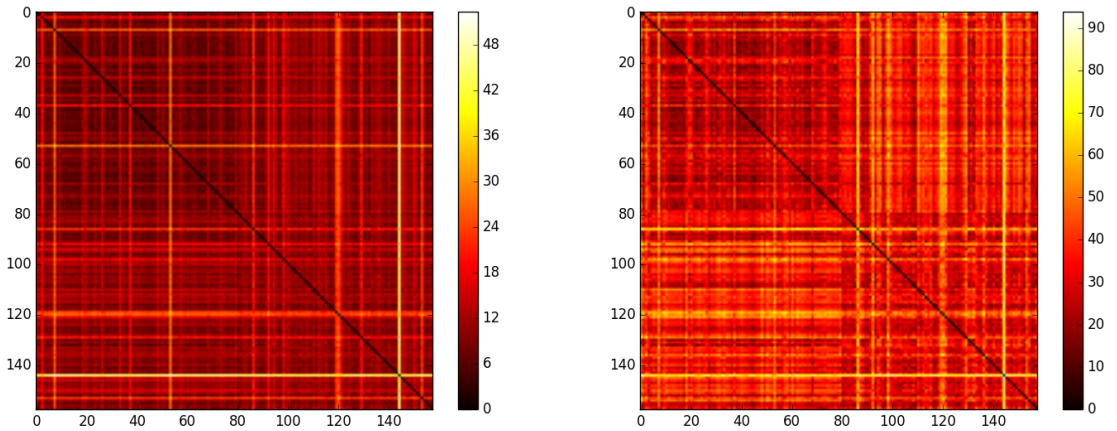
This fact is also illustrated in Figure 4.2 for the training set and in Figure 4.3 for the test set. These figures show the pairwise distance matrices between examples, both in the original space and in the embedding space. The matrices are ordered by classes. For the training set, the first 157 examples are from the "Pop_Rock" class and the last 157 examples are from the "Rap" class. For the test set it is the first 79 examples that are from the "Pop_Rock" class. The separation between the two classes is a little bit more visible in the embedding space. This is also the case for the examples in the test set (Figure 4.3). The examples from the first class are already close to each other in the original space, as it can observed by the "black" square in the top-left corner, but they stay close in the embedding space while the distance between them and the points from the other class increases.

The performance of the metric learning algorithm on the points of the test set are now analyzed because the objective at the end is the generalization of the learned metric to unknown data points. The previous section shows that MKPOE allows to increase the prediction accuracy of the kNN algorithm for the genre recognition task. It could be interesting to know on how many test points the learned metric has an impact. A quick test shows that in the case of the genre recognition, a better class is predicted for 16 test points (out of 158). Meanwhile, 10 correctly classified test points in the original space become misclassified in the embedding space. A good tuning of the hyperparameters allows a gain of 6 correctly classified points out of 158, which results in a increase of 4% of the accuracy. These results are consistent with those in the previous section.




(a) Original space (b) Embedding space

Figure 4.2: Genre recognition: pairwise distance matrix between examples of the training set in the original and the embedding space




(a) Original space (b) Embedding space

Figure 4.3: Genre recognition: pairwise distance matrix between examples of the test set in the original and the embedding space

### 4.4.2 Artist recognition

Now lets see how MKPOE performs on the dataset of the artist recognition task. In this case the analysis is reported for the 3_impostors set of distance constraints because it gives a better accuracy in less time, but the results obtained with the all_impostors set are very similar. Again, when $\lambda$ and the number of iterations are high enough, the number of satisfied constraints increases in the embedding space compared to the original space. Figure 4.4 shows that the number of distance constraints satisfied in the original space is very low compared to the number of constraints that MKPOE allows to satisfy in the embedding space.

When looking at the number of 3 nearest neighbors from the same class for every example of the training set, MKPOE performs well to bring examples of the same class closer to each other. The number of 3 nearest neighbors from the same class increased for 165 training points (out of 495) and decreased for only 27 training points. But the problem occurs when looking at the

points from the test set: for this fold, the accuracy drops from 0.242 without metric learning to 0.205 with MKPOE. So even if MKPOE gives nice results on the points from the training set, the learned metric doesn't generalize well to unseen points from the test set.

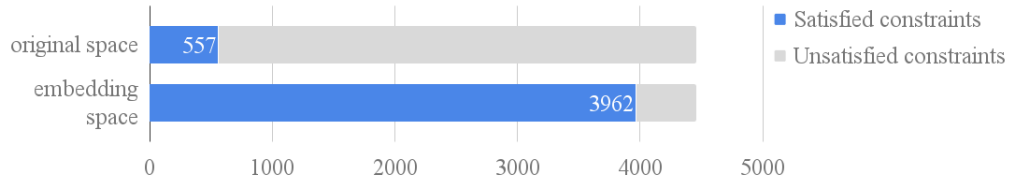
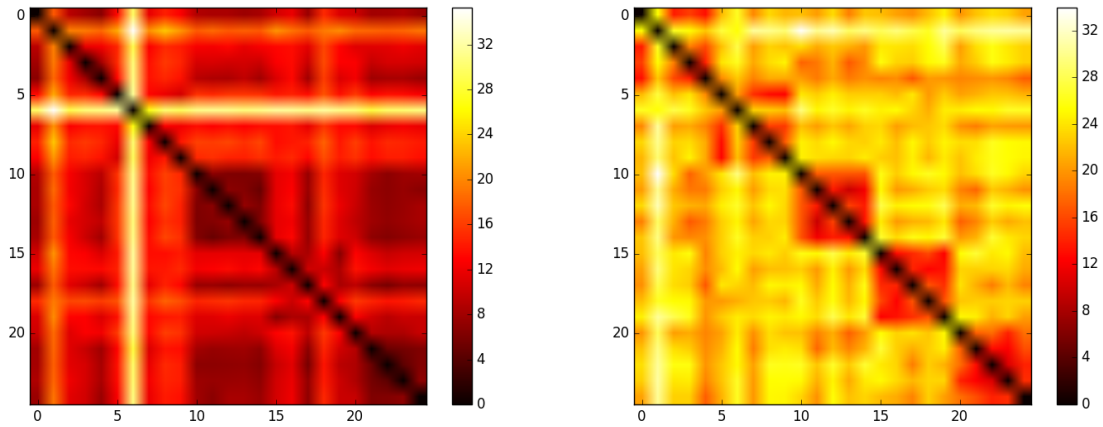



Figure 4.4: Artist recognition: number of satisfied constraints (from 3_impostors) in the embedding space compared to the original space. Total number of constraints: 4455

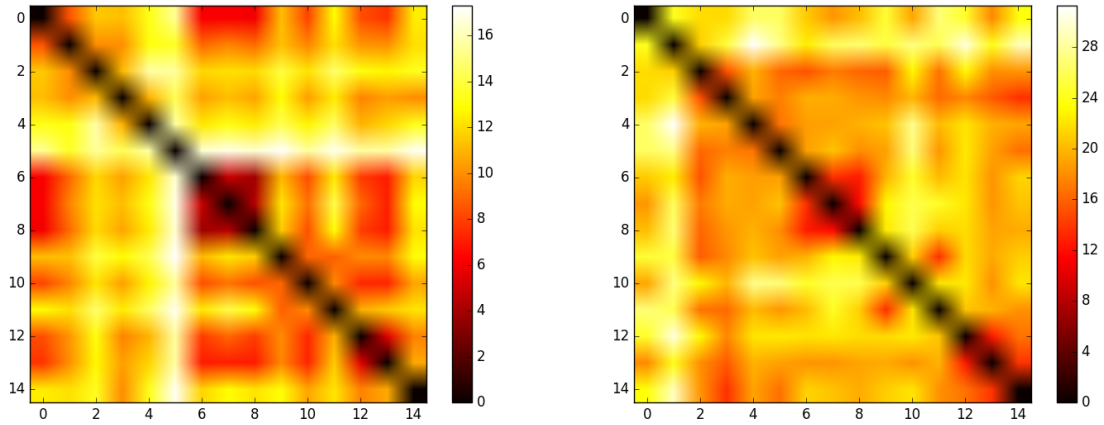
Figures 4.5 contains the ordered pairwise distance matrices between examples of the training set in the original space and in the embedding space obtained with MKPOE for the artist recognition task. The ordered pairwise matrices for the test set are illustrated in Figure 4.6. For a better visualisation, the number of classes has been limited to 5 (instead of 99) because each class contains a low number of examples: 5 examples per class in the training set and 3 examples per class in the test set. The separation between the classes is very clear in the embedding space of the training set. Indeed, the distances between points of the same class are much smaller than the distances between points from different classes. This confirms that MKPOE performs well to bring similar examples close to each other in the training set. But as illustrated in Figure 4.6, this doesn't generalize well to the examples of the test set.



(a) Original space

(b) Embedding space

Figure 4.5: Artist recognition: pairwise distance matrix between examples of the training set in the original and the embedding space (limited to 5 classes)



(a) Original space

(b) Embedding space

Figure 4.6: Artist recognition: pairwise distance matrix between examples of the test set in the original and the embedding space (limited to 5 classes)

## 4.5 Implementation

The implementation is done in Python and uses *scikit-learn*[1], an open source library for machine learning in Python. A nice feature of *scikit-learn* is the ability to create pipelines to sequentially apply a list of transformers, i.e. preprocessing steps that transform the feature representations (e.g., feature normalization or dimensionality reduction), followed by a final estimator (e.g., kNN). Multi-modal data can be processed in a pipeline thanks to `FeatureUnion`[2], which allows to apply different transformers to each modality of the data, for example a standardization step to some modalities and a tf-idf transformer for the lyrics. MKPOE is implemented as a transformer performing a metric learning preprocessing step.

Both the transformers and the final estimator could depend on hyperparameters that should be tuned, for example by cross-validation. With a pipeline, the hyperparameters of the whole pipeline can be tuned using `GridSearchCV`[3]. It optimizes the hyperparameters by doing a cross-validated exhaustive search over the hyperparameters' values specified in a parameter grid. The grid search can be parallelized by specifying a number of jobs to run in parallel. This is often helpful because the grid search can take a lot of time.

The hyperparameters to tune are:

- $\lambda$: the trade-off between regularization and constraint satisfaction. The set of tested values is $\lambda \in \{1, 10^3, 10^4, \ldots, 10^7\}$.
- $\gamma_1$: the hyperparameter of the RBF kernel for the *audio* modality. The set of tested values is $\gamma_1 \in \{10^{-3}, 10^{-2}, \ldots, 10\}$.
- $\gamma_2$: the hyperparameter of the RBF kernel for the third modality (*Other*). The set of tested values is the same as $\gamma_1$.
- $k$: the number of nearest neighbors for the kNN classifier. The range of tested values is $k \in \{1, 2, \ldots, 10\}$.

[1] `http://scikit-learn.org`

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.FeatureUnion.html`

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html`

In the current version of *scikit-learn* (version 0.18.1), `GridSearchCV` runs all the steps of the pipeline every time another combination of hyperparameters is tested. Algorithm 1 contains the pseudo-code of a cross-validation using `GridSearchCV`. As seen with the experimental results in the section 4.3, the metric learning step takes a lot of time (especially for MKPOE). The other preprocessing steps can take a lot of time too, it is the case for the generation of the triplet constraints. But often these preprocessing steps don't depend on some hyperparameters. For example, there is no need to regenerate all the triplet constraints and run MKPOE every time the hyperparameter $k$ changes. Even more, the generation of the triplet constraints as well as the standardization and tf-idf transformation of the lyrics don't depend on any hyperparameter, they just depend on the current training and test sets. However, the same split is performed for each hyperparameter combination.

**Algorithm 1** Cross-validation using GridSearchCV

```
for λ ∈ {1, 10^3, 10^4, ..., 10^7} do
    for γ1 ∈ {10^-3, 10^-2, ..., 10} do
        for γ2 ∈ {10^-3, 10^-2, ..., 10} do
            for k ∈ {1, 2, ..., 10} do
                for fold in split do
                    - Standardization
                    - Tf-idf transformation of the lyrics
                    - Generation of the triplet constraints
                    - Metric learning
                    - kNN prediction
                end for
                Get mean accuracy for this combination of hyperparameters
            end for
        end for
    end for
end for
```

In order to minimize the time spent on cross-validation, it is re-implemented to invert the loops for the hyperparameters and for the folds. The consequence of this inversion is that for each fold the accuracies obtained for each combination of the hyperparameters need to be stored (for example in a multi-dimensional array). But the time gained is worth this memory usage, especially when looking at the time taken by 1 run of MKPOE (see Table 4.1). Because this grid search is part of a nested cross-validation it is even more beneficial to minimize the number of calls to MKPOE when tuning the hyperparameters. The pseudo-code of the cross-validation implementation is shown in Algorithm 2.

Note that in order to reduce the total time taken by the cross-validation, in the implementation (Algorithm 2), the hyperparameter $\lambda$ is tuned sequentially compared to the other hyperparameters. It is thus not an exhaustive grid search but the other hyperparameters ($\gamma_1$, $\gamma_2$ and $k$) are fixed to reasonable values (i.e. values that gave good accuracies when tested with some values for $\lambda$) when tuning the hyperparameter $\lambda$.

The drawbacks of this approach is that the code is more complicated than a simple call to `GridSearchCV` and that the parallelisation of the search needs a bit more work instead of asking `GridSearchCV` to do the job (parallelisation is not used for this master's thesis).

**Algorithm 2** Implementation of cross-validation

Fix $\gamma_1$, $\gamma_2$, and $k$ to reasonable values
**for** fold in split **do**
    - Standardization
    - Tf-idf transformation of the lyrics
    - Generation of the triplet constraints
    **for** $\lambda \in \{1, 10^3, 10^4, \ldots, 10^7\}$ **do**
        - Metric learning
        - kNN prediction
    **end for**
**end for**
Fix $\lambda$ to best value found in the previous cross-validation
**for** fold in split **do**
    - Standardization
    - Tf-idf transformation of the lyrics
    - Generation of the triplet constraints
    **for** $\gamma_1 \in \{10^{-3}, 10^{-2}, \ldots, 10\}$ **do**
        **for** $\gamma_2 \in \{10^{-3}, 10^{-2}, \ldots, 10\}$ **do**
            Metric learning
            **for** $k \in \{1, 2, \ldots, 10\}$ **do**
                kNN prediction
            **end for**
        **end for**
    **end for**
**end for**
Get mean accuracy for each combination of the hyperparameters

## 4.6 Conclusion

A metric learning step allows to increase the performance of a kNN classifier for the genre recognition task. By taking advantage of the multi-modal aspect of the data, MKPOE is the approach that performs the best on this task. But for another task such as the artist recognition, the metric learning algorithm isn't able to learn a metric that generalizes well to unseen data. Despite the increase in accuracy for the genre recognition, the statistical tests reveal that the difference between the algorithms is not statistically significant (under the significance level 0.05). Further research could test these approaches on other datasets to confirm if there is a significant difference between them or not.

MKPOE allows an increase of accuracy for certain tasks but is computationally expensive. The next chapter explores a few approaches to reduce its execution time.

# Chapter 5

# How to scale MKPOE?

As seen in the previous chapter, MKPOE allows to increase the accuracy of a kNN algorithm on the genre recognition task. But the original dataset contains much more examples and the execution time of this algorithm significantly increases with the number of training examples. Indeed, having more examples implies more distance constraints and the execution time increases with the number of constraints. More examples also results in bigger kernel matrices because their size is $n \times n$ (n is the number of examples). Consequently, it takes more time to perform the operations on those kernels, especially matrix multiplication which is a frequent operation in MKPOE. The run time of this algorithm on a big dataset might be impractical. This chapters explores some approaches with the aim of reducing the execution time of MKPOE.

The first section is looking to an online adaptation of MKPOE, where only one pass on the set of distance constraints is needed. In the second section, the learned parameter matrices is restricted to be diagonal in order to decrease the execution time taken by the projection onto the PSD set. Always with the same goal, the third section applies a dimensionality reduction technique such that the parameter matrices are low-dimensional. Section 5.4 explores the gain obtained by a mini-batch version of the subgradient descent, where only a subset of the distance constraints is selected at each iteration. Section 5.5 is dedicated to adapt the formulation of MKPOE in order to optimize the metric learning problem using a Frank-Wolfe algorithm. Finally, section 5.7 contains the experimental results on the medium-scale datasets and the last section concludes this chapter.

## 5.1 Online approach

In order to process large-scale multi-modal data applications, Xia et al. propose Online Multi-modal Distance Learning (OMDL) [27], an online adaptation of MKPOE. The main difference with MKPOE is that only one pass on the set of triplet constraints $\mathcal{R}$ is performed, while MKPOE passes on all the constraints at each iteration.

At each iteration $t$, the algorithm processes a given triplet constraint $(x_i, x_j, x_k)$ and updates the metric for each modality $p$ by solving the following optimization problem:

$$\begin{aligned} \min_{W^p, \xi} \quad & \frac{1}{2}\|W^p - W^p_{t-1}\|^2_{\mathcal{F}} + \lambda_1 tr(W^p K^p) + \lambda_2 \xi \\ \text{s.t.} \quad & d^2_p(\phi_i, \phi_j) \doteq (K^p_i - K^p_j)^T W^p (K^p_i - K^p_j), \\ & d^2_p(\phi_i, \phi_j) + 1 \leq d^2_p(\phi_i, \phi_k) + \xi, \\ & \xi \geq 0, \\ & W^p \succeq 0 \end{aligned} \tag{5.1}$$

Problem (5.1) is based on the online passive-agressive learning algorithms [9]. At each iteration $t$, the problem is optimized such that a trade-off is done between having $W^p$ close to the previous parameter matrix $W^p_{t-1}$ ("conservativeness"), the regularization term $tr(W^p K^p)$ and minimizing the loss on the current triplet constraint: $\max\{0, d^2_p(\phi_i, \phi_j) - d^2_p(\phi_i, \phi_k) + 1\}$. The hyperparameters $\lambda_1$ and $\lambda_2$ control this trade-off.

The regularization term in problem (5.1) has been modified compared to the original version of the problem in [27] in order to have the same regularizer than in MKPOE. The third section of this chapter comes back to the true regularization term used by OMDL.

The solution to problem (5.1) can be solved by using:

$$W^p = W^p_{t-1} - \lambda_1 K^p - \tau^p K^p (E_{ij} - E_{ik}) K^p \tag{5.2}$$

where

$$\tau^p = \min\{\lambda_2, \frac{\ell(l_{t-1})}{\|K^p(E_{ij} - E_{ik})K^p\|^2_{\mathcal{F}}}\} \tag{5.3}$$

where $\ell(x) = \max\{0, x\}$ is the hinge loss function, $E_{ij} = (e_i - e_j)(e_i - e_j)^T$ and

$$l_{t-1} = -\lambda_1 tr(K^p K^p (E_{ij} - E_{ik}) K^p) + tr(W_{t-1} K^p (E_{ij} - E_{ik}) K^p) + 1 \tag{5.4}$$

and then by projecting $W^p$ onto the feasible set of PSD matrices. These equations are obtained based on the development in the original paper [27].

Because the parameters $W^p$ $(p = 1, ..., m)$ of the metric are optimized separately for each modality, they add a weight $\mu^p$ $(p = 1, ..., m)$ to each kernel when combining the modalities. This way the contribution of each modality to the learned metric in weighted. The embedding function is:

$$g(x) = (\sqrt{\mu^p} N^p K^p_x)^m_{p=1} \tag{5.5}$$

And the squared Euclidean distance in the embedding space is:

$$\begin{aligned} \|g(x_i) - g(x_j)\|^2_2 &= \sum_{p=1}^{m} \mu^p (K^p_i - K^p_j)^T (N^p)^T (N^p)(K^p_i - K^p_j) \\ &= \sum_{p=1}^{m} \mu^p d^2_p(\phi_i, \phi_j) \end{aligned} \tag{5.6}$$

The weights are updated at each iteration $t$ by following the idea of an online multiple kernel learning algorithm proposed in [15]:

$$\mu^p_t = \mu^p_{t-1} \ \eta^{z^p_t} \tag{5.7}$$

where $z^p_t$ takes value 1 if $d^2_p(\phi_i, \phi_j) > d^2_p(\phi_i, \phi_k)$ and 0 otherwise, and $\eta \in (0, 1)$ is a hyperparameter giving the factor by which the weight is decreased when $z^p_t$ is 1. So the weight assigned to a modality will be small if the distance constraints are not often satisfied in this modality.

This algorithm has two set of variables to learn: the parameter matrices $W^p$ $(p = 1, ..., m)$ of the learned metric and the weights $\mu^p$ $(p = 1, ..., m)$ assigned to each kernel when combining the modalities. The full OMDL algorithm is shown in Algorithm 3.

Table 5.1 compares the execution time of the OMDL algorithm with the execution time of MKPOE. Unfortunately, OMDL takes much more time than MKPOE, especially when the number of constraints is high. In this case OMDL is not practical. The rest of this chapter explores other techniques to reduce the execution time of MKPOE. Some are applied to OMDL to see if the execution time of OMDL can fall below that of MKPOE.

**Algorithm 3** Online Multi-modal Distance Learning (OMDL) [27]

**Input:**

- training set $\mathcal{X} = \{x_i | i = 1, ..., n\}$
- $m$ kernel matrices $K^p \ \forall p \in \{1, ..., m\}$
- trade-off parameters $\lambda_1 > 0, \lambda_2 > 0$
- discount weight $\eta \in (0, 1)$

**Output:** $W^p, \mu^p \ \forall p \in \{1, \ldots, m\}$

```
Initialization: W_0^p = I, μ_0^p = 1 ∀p ∈ {1, ..., m}
for t = 1, 2, ..., T do
    receive a triplet (x_i, x_j, x_k) from R
    for p = 1,..., m do
        compute τ_t^p with (5.3)
        compute W_t^p with (5.2)
        PSD projection: λ_i → max(0, λ_i) ∀i ∈ {1, ..., n}
        if d_p^2(φ_i, φ_j) > d_p^2(φ_i, φ_k) then
            z_t^p = 1
        else
            z_t^p = 0
        end if
        update μ_t^p = μ_{t-1}^p η^{z_t^p}
    end for
end for
```

| | 3_impostors | all_impostors |
|---|---|---|
| MKPOE | 43s | 4min20 |
| OMDL | 2min26 | 2h09 |

Table 5.1: Execution time of MKPOE and OMDL on the dataset of the genre recognition task

## 5.2 Diagonal matrices

A profiling tool shows that the eigendecomposition of the matrices $W^p$ is the step that takes the most time in both algorithm: 51.16% of the execution time for MKPOE and 75.37% for OMDL. This eigendecomposition is needed in order to project the matrices $W^p$ on the set of positive semi-definite matrices, which is performed at the end of each iteration for both algorithms. It has a time complexity of $O(n^3)$ (where $n$ is the size of the training set). Reducing the time taken by the projection step is useful to reduce the execution time of these algorithms.

As proposed in [19], diagonal matrices $W^p$ can be learned in order to reduce the computational cost of MKPOE. Since the eigenvalues of a diagonal matrix are the element of its diagonal and a matrix is positive semi-definite if its eigenvalues are positive, the projection of a diagonal matrix is simply done by setting all the negative elements of its diagonal to 0. The time complexity of this projection is thus $O(n)$ for each modality instead of $O(n^3)$.

At each iteration of MKPOE, every distance constraint needs to be verified if satisfied or not. The distance between two points in the embedding space can be computed directly by the squared Mahalanobis distance:

$$d^2(\phi_i, \phi_j) = \sum_{p=1}^{m} (K_i^p - K_j^p)^T W^p (K_i^p - K_j^p) \tag{5.8}$$

Or the distance can be computed by projecting the data points in the embedding space at each iteration and using the Euclidean distance between points in this new space. Computing the distance in the first case has a time complexity of $O(n^2)$ while the time complexity for computing the Euclidean distance is $O(n)$ but the eigendecompositions $W^p = V_p \Lambda_p V_p^T$ are needed to embed the data points in the new space. Indeed, the embedding function is:

$$g(x) = (N^p K_x^p)_{p=1}^m \tag{5.9}$$

where each $N^p$ is obtained with $N^p = \Lambda_p^{1/2} V_p^T$. It is often the case that the number of distance constraints is high (higher than $n$, the size of the training set) and in this case the experiments empirically showed that it is more efficient to compute the distance by first embedding the points and then using the Euclidean distance.

An eigendecomposition of $W^p$ ($\forall p \in \{1, \ldots, m\}$) at each iteration of MKPOE is still necessary to embed the data points in the new space. This eigendecomposition is very efficient when $W^p$ is diagonal. Indeed, if $W^p$ is diagonal then $V_p = V_p^T = I_n$ (the identity matrix of size $n$) and $\Lambda_p = W^p$. So $N^p$ is simply:

$$N^p = \Lambda_p^{1/2} V^T = (W^p)^{1/2} \tag{5.10}$$

which can be computed in $O(n)$.

For OMDL it is not necessary to embed the data points at each iteration because only one distance constraint is processed at each iteration. So for OMDL the distance between two points is computed in the embedding space using the squared Mahalanobis distance in (5.8).

Table 5.2 compares the execution time of MKPOE-diag and OMDL-diag (i.e. MKPOE and OMDL with diagonal matrices) when using the 3_impostors and the all_impostors sets of constraints. As expected, there is a great difference in the execution time between the diagonal versions of these algorithms and the original versions (see Table 5.1), especially for OMDL, for which the projection step is called many times.

| | 3_impostors | all_impostors |
|---|---|---|
| MKPOE-diag | 11.3s | 3min02 |
| OMDL-diag | 13s | 10min57 |

Table 5.2: Execution time of MKPOE-diag and OMDL-diag on the dataset of the genre recognition task

However, MKPOE-diag is again faster than OMDL-diag, so only the accuracy obtained with MKPOE-diag is considered because the latter is expected to give a worse accuracy. Moreover, the total time of the nested cross-validation will be longer for OMDL-diag than MKPOE-diag because of two more hyperparameters to tune ($\lambda_2$ and $\eta$).

The mean accuracy of a nested stratified 3-fold cross-validation is 0.818 when using the 3_impostors set of constraints and 0.852 when using the all_impostors set of constraints. In both cases, the accuracy is under the estimated accuracy of kNN without metric learning (0.865). Figures 5.1 and 5.2 show that the number of satisfied constraints on the training set in the embedding space is smaller than in the original space. The diagonal version of MKPOE fails to learn a performing metric.

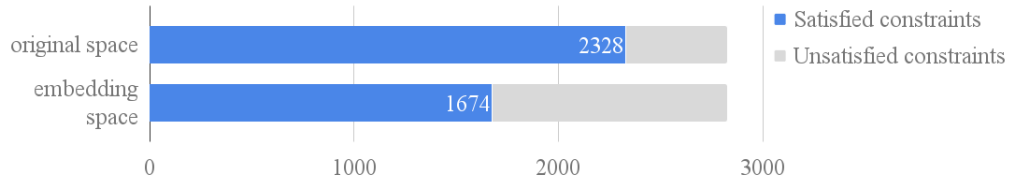



Figure 5.1: Genre recognition: number of satisfied constraints (in 3_impostors) for a diagonal version of MKPOE. Total number of constraints: 2826

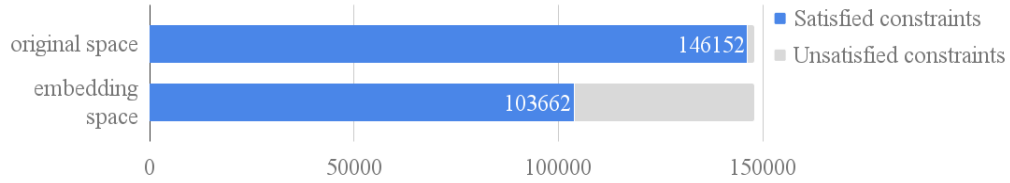



Figure 5.2: Genre recognition: number of satisfied constraints (in all_impostors) for a diagonal version of MKPOE. Total number of constraints: 147894

## 5.3 Dimensionality reduction

A frequent method to decrease the computation cost of metric learning algorithms is to preprocess the data with a dimensionality reduction technique [4]. For example, the authors of LMNN use PCA in the experiments of their paper [24]. In the nonlinear case, the distance

$$d(\phi_i, \phi_j) = \sqrt{(K_i - K_j)^T W (K_i - K_j)} \tag{5.11}$$

can be seen as a Mahalanobis distance between the vectors $K_i$ and $K_j$ where $W$ is the parameter matrix to learn. Dimensionality reduction techniques can be applied to the data contained in the matrix $K \in \mathbb{R}^{n \times n}$ by considering the columns of $K$ as data points. This way, the dimensions of the new matrix representing the data $K_{LR}$ are $n' \times n$ ($n' < n$). The dimensions of the parameter matrix are $n' \times n'$, so the eigendecomposition is much faster.

In OMDL-LR [27], the authors propose to use random projection to decrease the dimensions of $W$, by following the idea in [6]. In random projection, the original $n$-dimensional data is projected to a $n'$-dimensional subspace by using a random matrix $P \in \mathbb{R}^{n' \times n}$. The elements of $P$ are generated at random following a Gaussian distribution and in such way that the rows of $P$ are unit vectors orthogonal to each other. The matrix $K_{LR} \in \mathbb{R}^{n' \times n}$ containing the low-dimensional data is obtained with:

$$K_{LR} = PK \tag{5.12}$$

A low-rank approximation of $W$ is obtained with $P^T W_{LR} P$.

The idea behind random projection comes from the Johnson-Lindenstrauss lemma [16], which states that points in a high-dimensional space can be projected into a randomly selected subspace such that the distance between the points are approximately preserved. Random projection is probably used in OMDL because it is a simple and fast dimensionality reduction technique. It is important to use a fast technique because the goal is to decrease the computational cost of the metric learning phase.

In the multi-modal case, a random projection matrix $P^p$ is generated for each modality $p$ in order to decrease the dimensions of each $W^p$. It is not obvious how to adapt the regularization term used by MKPOE in order to use the low-dimensional $W^p_{LR} \in \mathbb{R}^{n' \times n'}$. But as stated earlier, the

authors of OMDL also change the regularization term in order to improve the effectiveness of the method. In addition to the likely increase in accuracy, a key advantage of this new regularizer is its ability to use the low-dimensional matrices $W^p_{LR}$.

They propose to use the graph Laplacian regularizer:

$$tr(V\mathcal{L}V^T) \tag{5.13}$$

where $V = [g(x_1), \ldots, g(x_n)]$ is a matrix containing the embeddings of the training points $x_i$ as columns and $\mathcal{L}$ is the symmetric normalized Laplacian matrix.

**Definition 3.** The *symmetric normalized Laplacian matrix* is defined by:

$$\mathcal{L} = I - D^{-1/2}SD^{-1/2} \tag{5.14}$$

where $S \in \mathbb{R}^{n\times n}$ is a symmetric similarity matrix where each entry $S_{ij}$ is the similarity between data points $x_i$ and $x_j$ in the original space and $D$ is a diagonal matrix with diagonal elements defined as $D_i = \sum_{j=1}^{n} S_{ij}$.

The similarity matrix $S$ is defined as:

$$S_{ij} = \begin{cases} k(x_i, x_j) & \text{if } x_i \text{ or } x_j \text{ is in the } k'\text{-neighborhood of the other one} \\ 0 & \text{otherwise} \end{cases} \tag{5.15}$$

where $k'$ is fixed to 5 and $k(x_i, x_j)$ is the value of the kernel between $x_i$ and $x_j$. The goal of this regularizer is to enforce small Euclidean distances in the embedding space between similar points (in the sense of the similarity matrix).

Thanks to $V = NK$, the regularizer can be written in terms of $W$ and $K$:

$$tr(V\mathcal{L}V^T) = tr(V^TV\mathcal{L}) = tr((NK)^TNK\mathcal{L}) = tr(K^TN^TNK\mathcal{L}) = tr(K^TWK\mathcal{L}) \tag{5.16}$$

For the multi-modal case, the regularization term is:

$$\sum_{p=1}^{m} tr((K^p)^TW^pK^p\mathcal{L}^p) \tag{5.17}$$

Changing the regularization term in OMDL (problem (5.1)) gives this update rule:

$$W^p = W^p_{t-1} - \lambda_1 K^p\mathcal{L}^p(K^p)^T - \tau^p K^p(E_{ij} - E_{ik})(K^p)^T \tag{5.18}$$

where

$$\tau^p = \min\{\lambda_2, \frac{h(l_{t-1})}{\|K^p(E_{ij} - E_{ik})(K^p)^T\|^2_{\mathcal{F}}}\} \tag{5.19}$$

and

$$l_{t-1} = -\lambda_1 tr(K^p\mathcal{L}^p(K^p)^TK^p(E_{ij} - E_{ik})(K^p)^T) + tr(W_{t-1}K^p(E_{ij} - E_{ik})(K^p)^T) + 1 \tag{5.20}$$

$K^p$ and $W^p$ can easily be substituted to use the low-dimensional $K^p_{LR}$ and $W^p_{LR}$.

Nevertheless, the authors of OMDL didn't try MKPOE with the graph Laplacian regularizer and the low-dimensional matrices $W^p_{LR}$. It is straightforward to adapt MKPOE in problem (2.16) to use the regularization term (5.17) and the low-dimensional matrices $K^p_{LR}$ and $W^p_{LR}$.

The previous section shows that diagonally-constrained $W^p$ is failing to learn a good metric for this problem. Using low-dimensional matrices $W^p_{LR} \in \mathbb{R}^{n' \times n'}$ (such that $n' < n$) allows to speed up the execution time of the algorithm while maintaining good performance. Since most of OMDL's execution time comes from the eigendecomposition of $W^p$, the performances of the low-dimensional versions of both MKPOE and OMDL are analyzed next.

Table 5.3 compares the performance of OMDL-LR and MKPOE-LR (i.e. MKPOE with dimensionality reduction and the regularizer of OMDL-LR). In all cases, the accuracy obtained is better than the accuracy obtained by kNN without metric learning. OMDL-LR is still slower than MKPOE-LR. Only the accuracies obtained with MKPOE-LR are considered because OMDL-LR is expected to give worse accuracies and take much more time during the nested cross-validation because of the larger number of hyperparameters to tune. As before, there is a slight difference between the accuracies of MKPOE-LR with 3_impostors and all_impostors, but the time taken when using the 3_impostors set of constraints is much smaller. It could be preferable to use this set in practice. There is a trade-off between performance and execution time when choosing the dimension $n'$ of the low-dimensional space. As shown in Table 5.3, the difference in accuracy is quit high between $n' = 50$ and $n' = 100$ so it might not be a good idea to choose a too small $n'$.

| | $n'$ | accuracy | SD | time (1 run) | total time |
|---|---|---|---|---|---|
| MKPOE-LR 3_impostors | 50 | 0.871 | 0.021 | **8.2s** | 30min58 |
| | 100 | 0.892 | 0.013 | 12s | 45min06 |
| OMDL-LR 3_impostors | 50 | - | 13.5s | - | |
| MKPOE-LR all_impostors | 50 | 0.867 | 0.049 | 2min35 | 5h39 |
| | 100 | **0.894** | 0.016 | 2min51 | 6h37 |
| OMDL-LR all_impostors | 50 | - | 11min15 | - | |

Table 5.3: Performance of MKPOE-LR and OMDL-LR on the genre recognition dataset. $n'$ is the dimension of the low-dimensional space (SD = standard deviation)

## 5.4 Mini-batch gradient descent

Besides the eigendecomposition needed to enforce the PSD constraint after each iteration, the subgradient descent has to pass over all the distance constraints at each iteration to compute the gradient. If the number of distance constraints is high, the computation of the gradient might take some time. As proposed in [18] and [20], a mini-batch approximation of the gradient can be used by randomly selecting a subset $\mathcal{R}'$ of constraints at each iteration, with $|\mathcal{R}'| \ll |\mathcal{R}|$.

First, the size of the subset of constraints needs to be chosen. In the experiments, the size of $\mathcal{R}'$ is fixed to 20% of $|\mathcal{R}'|$.

In order to use the "bold driver" method to update the learning rate (see section 3.7), the exact cost of the objective function needs to be computed at each iteration, which requires a pass over all the constraints. Because of that, the mini-batch gradient descent is useless if the "bold driver" method is used. Therefore, a geometric decrease of the learning rate is used in the experiments of this section.

Table 5.4 reports the performance obtained by MKPOE when using a mini-batch gradient descent by randomly selecting 20% of the constraints at each iteration. As expected, the execution time of the mini-batch version is smaller than the execution time of the batch gradient descent without a big loss in accuracy. As a reminder, the results of MKPOE reported in Table 4.2 have an accuracy of 0.894 for an execution time of 43s for the 3_impostors set, and an accuracy of 0.905

| | accuracy | SD | time (1 run) | total time |
|---|---|---|---|---|
| 3_impostors | 0.89 | 0.028 | 34.5s | 1h19 |
| all_impostors | 0.877 | 0.024 | 2min03 | 4h38 |

Table 5.4: Performance of MKPOE with a mini-batch gradient descent on the genre recognition dataset when 20% of the constraints are chosen at each iteration (SD = standard deviation)

for an execution time of 4min20 for the all_impostors set. MKPOE-LR performs better in terms of accuracy (when $n' = 100$) for a comparable decrease in the execution time, but if needed the two methods can be used together.

It should be pointed out that the performance of the mini-batch version of MKPOE could be better if an optimal learning rate is chosen. The learning rate is fixed following a geometric progression, i.e. $\alpha_t = s \cdot c^{t+1}$ where $s$ is a scaling parameter fixed to 0.1 in the experiments and $c = 0.99$. But in this case the learning rate often decreases too quickly. For example, in Figure 5.3, the cost obtained with a learning rate following a geometric progression converges quickly, while with a constant step size (fixed to 0.7 in this case), the cost doesn't converge after 400 iterations. Adapting the scaling parameter $s$ and the decreasing parameter $c$ for every combination of hyperparameters is not an easy task.

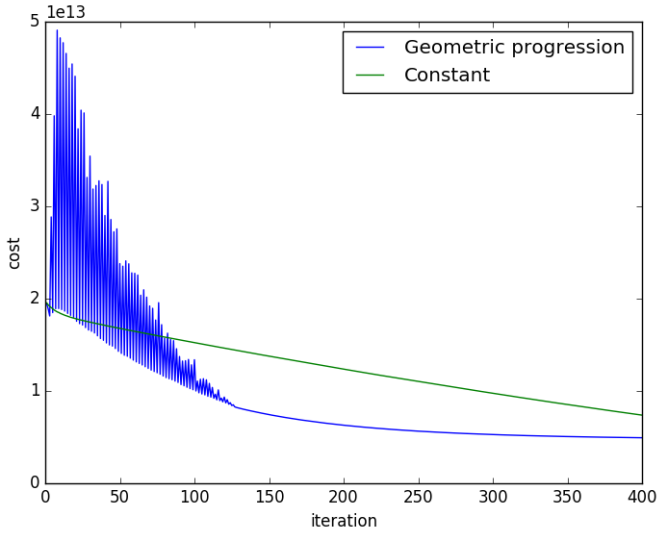



Figure 5.3: Convergence of MKPOE with a mini-batch gradient descent when using the all_impostors set of constraints

## 5.5 Frank-Wolfe optimization

An efficient way to enforce the PSD constraints is to learn matrices $W^p$ such that they are always in the set of PSD constraints. This avoids the costly projection onto the feasible set at the end of each iteration. In [18], Liu et al. propose to learn the parameter matrix as a convex combination of rank-one 4-sparse base matrices by using a Frank-Wolfe optimization algorithm.

In their paper, they learn a linear metric with a parameter matrix $M$ of size $d \times d$ (where $d$ is the dimensionality of the data). The 4-sparse matrices of size $d \times d$ are chosen from the set

$\mathcal{B}_\lambda = \cup_{ij}\{P_\lambda^{(ij)}, N_\lambda^{(ij)}\}$ for any pairs of features $i, j \in \{1, ..., d\}, i \neq j$ where

$$P_\lambda^{(ij)} = \lambda(e_i + e_j)(e_i + e_j)^T = \begin{pmatrix} . & . & . & . & . \\ . & \lambda & . & \lambda & . \\ . & . & . & . & . \\ . & \lambda & . & \lambda & . \\ . & . & . & . & . \end{pmatrix}$$

$$N_\lambda^{(ij)} = \lambda(e_i - e_j)(e_i - e_j)^T = \begin{pmatrix} . & . & . & . & . \\ . & \lambda & . & -\lambda & . \\ . & . & . & . & . \\ . & -\lambda & . & \lambda & . \\ . & . & . & . & . \end{pmatrix}$$

and $\lambda > 0$ is a scale hyperparameter. It is easy to prove that the base matrices are PSD. Since $M$ is a convex combination of PSD matrices, it is also a PSD matrix. The $O(d^3)$ time cost of the projection onto the feasible set is avoided.

The authors propose to use the Frank-Wolfe alorithm [12] to optimize the problem. The Frank-Wolfe algorithm is an iterative algorithm to minimize a convex and continuously differentiable function over a compact convex set. At each iteration, the algorithm considers a linear approximation of the objective function to minimize and moves in the direction minimizing this linear approximation while staying in the feasible domain. The minimizer of the linear approximation is a vertex of the feasible domain and each iterate is a sparse convex combination of the vertices of the domain.

In the paper, the objective function to minimize is the average loss over the triplet constraints in $\mathcal{R}$:

$$f(M) = \frac{1}{|\mathcal{R}|} \sum_{(x_i, x_j, x_k) \in \mathcal{R}} \ell(S_M(x_i, x_j) - S_M(x_i, x_k)) \tag{5.21}$$

where $S_M(x_i, x_j)$ is a similarity function between data points $x_i$ and $x_j$ and, in this case, $\ell(x)$ is the smoothed hinge loss:

$$\ell(x) = \begin{cases} \frac{1}{2} - x & \text{if } x \leq 0 \\ \frac{1}{2}(1 - x)^2 & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x \geq 1 \end{cases} \tag{5.22}$$

The smoothed hinge loss is used instead of the hinge loss because it is differentiable. No regularization term is necessary because overfitting is avoided by limiting the number of base matrices used in $M$.

At each iteration, the matrix $M$ is represented as a convex combination of the base matrices:

$$M = \sum_{B \in \mathcal{B}_\lambda} \gamma_B B, \quad \text{where} \quad \sum_{B \in \mathcal{B}_\lambda} \gamma_B = 1, \ \gamma_B \geq 0 \quad \forall B \in \mathcal{B}_\lambda \tag{5.23}$$

The next subsection explains how to adapt MKPOE to use the Frank-Wolfe algorithm based on the formulation proposed in this section. This means adapting the formulation to a nonlinear and multi-modal approach and learning a Mahalanobis distance function instead of a similarity function.

### 5.5.1 Adapting MKPOE

As stated in [19], the learned squared Mahalanobis distance between $\phi_i$ and $\phi_j$ can be expressed as:

$$\begin{aligned} d^2(\phi_i, \phi_j) &= \sum_{p=1}^{m} (K_i^p - K_j^p)^T W^p (K_i^p - K_j^p) \\ &= \sum_{p=1}^{m} (e_i - e_j)^T (K^p)^T W^p K^p (e_i - e_j) \\ &= \sum_{p=1}^{m} tr((K^p)^T W^p K^p (e_i - e_j)(e_i - e_j)^T) = \sum_{p=1}^{m} tr(W^p K^p E_{ij} (K^p)^T) \\ &= \sum_{p=1}^{m} \langle W^p, K^p E_{ij} (K^p)^T \rangle_{\mathcal{F}} \end{aligned} \tag{5.24}$$

where $e_i$ is the $i^{\text{th}}$ standard basis column vector, $E_{ij} = (e_i - e_j)(e_i - e_j)^T$ and $\langle \cdot, \cdot \rangle_{\mathcal{F}}$ is the Frobenius inner product.

**Definition 4.** The *Frobenius inner product* of two real-valued $n \times m$ matrices $A$ and $B$ is defined by

$$\langle A, B \rangle_{\mathcal{F}} = \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} B_{ij} = tr(A^T B) \tag{5.25}$$

The distance constraints with a margin of 1 can be written as:

$$\begin{aligned} & d^2(\phi_i, \phi_j) + 1 \leq d^2(\phi_i, \phi_k) \\ \Leftrightarrow & \sum_{p=1}^{m} \langle W^p, K^p E_{ij} (K^p)^T \rangle_{\mathcal{F}} + 1 \leq \sum_{p=1}^{m} \langle W^p, K^p E_{ik} (K^p)^T \rangle_{\mathcal{F}} \\ & \Leftrightarrow 1 \leq \sum_{p=1}^{m} \langle W^p, K^p (E_{ik} - E_{ij}) (K^p)^T \rangle_{\mathcal{F}} \end{aligned} \tag{5.26}$$

The objective function in (5.21) is adapted to obtain the objective function to minimize:

$$f(W^1, \ldots, W^m) = \frac{1}{|\mathcal{R}|} \sum_{(x_i, x_j, x_k) \in \mathcal{R}} \ell\Big(\sum_{p=1}^{m} \langle W^p, K^p (E_{ik} - E_{ij}) (K^p)^T \rangle_{\mathcal{F}}\Big) \tag{5.27}$$

where $|\mathcal{R}|$ is the number of triplet constraints and $\ell(x)$ is the smoothed hinge loss (5.22).

The gradient of the objective function is composed of the partial derivative of each modality:

$$\frac{\partial f}{\partial W^p} = \frac{1}{|\mathcal{R}|} \sum_{(x_i, x_j, x_k) \in \mathcal{R}} G^p_{(x_i, x_j, x_k)} \Big(\sum_{p=1}^{m} \langle W^p, K^p (E_{ik} - E_{ij}) (K^p)^T \rangle_{\mathcal{F}}\Big) \tag{5.28}$$

with

$$G^p_{(x_i, x_j, x_k)}(x) = \begin{cases} -\frac{\partial x}{\partial W^p} & \text{if } x \leq 0 \\ -(1 - x) \frac{\partial x}{\partial W^p} & \text{if } 0 \leq x \leq 1 \\ 0 & \text{if } x \geq 1 \end{cases} \tag{5.29}$$

where $\frac{\partial x}{\partial W^p} = K^p (E_{ik} - E_{ij}) (K^p)^T$ when $x = \sum_{p=1}^{m} \langle W^p, K^p (E_{ik} - E_{ij}) (K^p)^T \rangle_{\mathcal{F}}$.

Each $W^p$ is a convex combination of base matrices (as matrix $M$ in (5.23)). In this case the base matrices are of size $n \times n$ and represent pairs of data points instead of pairs of features.

### 5.5.2 Frank-Wolfe algorithm

At each iteration $t$ of the Frank-Wolfe algorithm, the matrices $W^p$ are updated with

$$W^p_{t+1} = W^p_t + \alpha_t(B^p_t - W^p_t) \tag{5.30}$$

where $\alpha_t$ the step size at iteration $t$ and $B^p_t$ is the base matrix minimizing the linear approximation given by the first-order Taylor approximation of $f$ around $W_t = (W^1_t, \dots, W^m_t)$:

$$B^p_t = \underset{B \in \mathcal{B}_\lambda}{\operatorname{argmin}} \langle B, \frac{\partial f}{\partial W^p}(W_t) \rangle_{\mathcal{F}} \tag{5.31}$$

The matrix $B$ is 4-sparse so the Frobenius inner product can be computed very quickly by summing the terms obtained for the 4 non-zero entries of $B$.

To avoid testing all pairs $(i, j)$ when searching for the best $B$, the authors propose to use a fast-heuristic by sequentially finding the best $j$ with $i$ fixed at random, solving the problem with $B \in \cup_j \{P^{(ij)}_\lambda, N^{(ij)}_\lambda\}$, and then finding the best $i$ by fixing $j$ to the best value found.

As with the gradient descent, the step size $\alpha_t$ needs to be chosen at each iteration $t$. For the Frank-Wolfe algorithm, it usually follows a decreasing function such as $\alpha_t = \frac{2}{t+2}$ [13]. A frequent improvement is to do a line-search to find the optimal step size in the current direction by solving the following minimization problem:

$$\alpha_t = \underset{\alpha \in [0,1]}{\operatorname{argmin}} f((1-\alpha)W^p_t + \alpha B^p_t) \tag{5.32}$$

But this is too computationally expensive because the gradient of $f$ with respect to $\alpha$ must be computed at each iteration of the line-search, which requires a pass on all the triplet constraints at each iteration to known if they are satisfied or not.

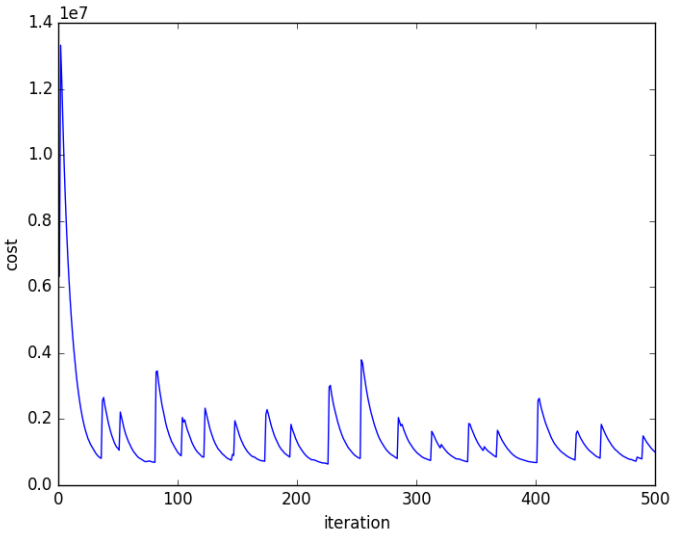



Figure 5.4: Convergence of the Frank-Wolfe algorithm when using the 3_impostors set of constraints and with the learning rate following a geometric progression

It is hard to learn a good metric with this technique. In each experiment, the convergence was reached too early, even when changing the decreasing function of $\alpha_t$ with a geometric progression in order to have a slower decrease of the step size. Indeed, the number of unsatisfied constraints

when the convergence is reached is always higher than the number of unsatisfied constraints when using the Euclidean distance in the original space (i.e. without a metric learning step). And the performance of the learned metric is worse than using the Euclidean distance without a metric learning step, both for the training set and for the test set.

Figure 5.4 illustrates the convergence of the Frank-Wolfe algorithm when the learning rate follows a geometric progression. The cost of the objective function decreases but, as it is observed in the experiments, the convergence is reached too early.

## 5.6 Experiments on medium-scale datasets

MKPOE-LR with the 3_impostors set of distance constraints is the method that gives the best trade-off between accuracy and execution time. This section assesses the performance of MKPOE-LR on two medium-scale datasets. Both datasets contain all the genres for which there is more than 100 songs. This means that there are 10 classes (see Table 3.1 on page 16 for the distribution of the data points in the classes). The first dataset is balanced and contains exactly 120 randomly sampled songs per genre. The second one contains all the songs from these genres, resulting in a imbalanced and larger dataset.

For these experiments, the dimension of the low-dimensional space is fixed to $n' = 200$. The set of distance constraints used is 3_impostors, meaning that 9 triplet constraints are generated for each data point (because 3 target neighbors and 3 impostors are selected). And the maximum number of iterations allowed is increased to 1500, because experimental observations show that 400 iterations are not enough for this setting in some cases.

### 5.6.1 Balanced medium-scale dataset

This dataset has 1200 data points. The accuracies obtained by a kNN classifier without a metric learning step and when MKPOE-LR is used are reported in Table 5.5. MKPOE-LR allows an increase in accuracy of 2.5%, which is not super high but is obtained in a reasonable amount of time. This accuracy is obtained by a 3-fold nested-cross validation, whose total time is reported under the column "total time".

| | accuracy | SD | time (1 run) | total time |
|---|---|---|---|---|
| No metric learning | 0.34 | 0.005 | 9ms | 3s |
| MKPOE-LR | **0.365** | 0.02 | 2min42 | 9h05 |

Table 5.5: Performance of MKPOE-LR on the balanced medium-scale dataset ($n' = 200$) (SD = standard deviation)

Figure 5.5 contains an analysis of the number of satisfied constraints before and after the metric learning step on the first fold of the cross-validation. MKPOE-LR can increase the number of satisfied constraints. On the first fold, it results in an increase in prediction accuracy of 6.75% for the training set and 5% for the test set.

### 5.6.2 Full medium-scale dataset

This dataset has 3520 data points. Table 5.6 compares the performance of a kNN classifier without metric learning and with MKPOE-LR. This time the increase in accuracy is only of 0.4%. Again, the accuracy is obtained by a 3-fold nested-cross validation. The dimensionality reduction is important to reduce the execution time of MKPOE, because MKPOE-LR with 1500 iterations has an execution time of 7min19 while MKPOE with 400 iterations took 1h18 to run.

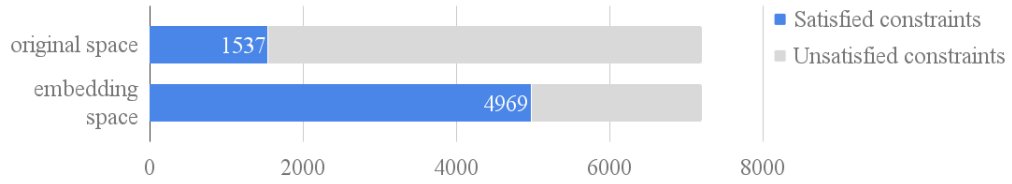



Figure 5.5: Genre recognition: number of satisfied constraints (from 3_impostors) for the balanced medium-scale dataset. Total number of constraints: 7200

| | accuracy | time (1 run) | total time | |
|---|---|---|---|---|
| No metric learning | 0.591 | 0.003 | 0.03s | 21.5s |
| MKPOE-LR | **0.595** | 0.018 | 7min19 | 21h32 |

Table 5.6: Performance of MKPOE-LR on the full medium-scale dataset ($n' = 200$)

Again, an analysis is performed on the first fold of the cross-validation. The number of satisfied constraints before and after the metric learning step on the first fold are reported in Figure 5.6. MKPOE-LR increases the number of satisfied constraints on the first fold by 4131, in other words around 20% more distance constraints are satisfied thanks to MKPOE-LR. In the experiments, the accuracy on the training set is always increased thanks to MKPOE, but for some folds the accuracy on the test set is worse than without metric learning. This means that the learned metric doesn't always generalize well to unseen data.

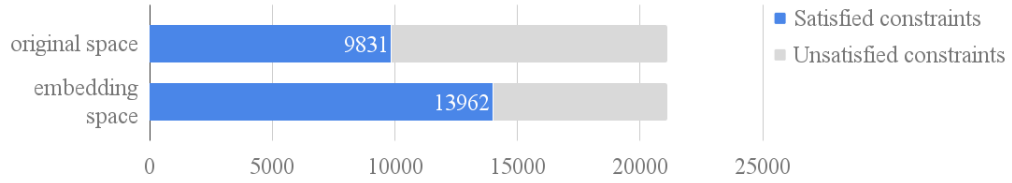



Figure 5.6: Genre recognition: number of satisfied constraints (from 3_impostors) for the full medium-scale dataset. Total number of constraints: 21105

Note that because this dataset is imbalanced and the "Pop_Rock" class dominates the other classes (54.7% of the dataset), the accuracy might not be the best measure to assess the performance. Indeed, an accuracy of 0.547 can already be reached by always predicting the class "Pop_Rock". With a imbalanced dataset it is more meaningful to look at the confusion matrix, where correct predictions are on the diagonal of the table and where prediction errors are outside the diagonal. The confusion matrix in Table 5.7 is obtained with the predictions on the test set of the first fold. It shows that a large portion of the points are predicted in the "Pop_Rock" class, even if it is not the correct class. But in general, after the "Pop_Rock" class, the second most predicted value for each class is the correct class, as illustrated by high values on the diagonal.

MKPOE-LR allows to increase the accuracy for both medium-scale datasets but the magnitude of the increase is not very high, especially on the full dataset. Depending on the time available, it could be judicious to learn a better metric, for example by increasing the dimensions of the low-dimensional space or by learning on a larger set of distance constraints, at the cost of a longer training time.

| | Blues | Country | Elec. | Int. | Jazz | Latin | Pop_Rock | Rap | Reggae | RnB |
|---|---|---|---|---|---|---|---|---|---|---|
| Blues | 3 | 6 | 0 | 1 | 3 | 3 | 26 | 0 | 0 | 0 |
| Country | 0 | 14 | 1 | 2 | 0 | 2 | 39 | 1 | 0 | 0 |
| Elec. | 0 | 1 | 10 | 2 | 0 | 4 | 54 | 3 | 0 | 2 |
| Int. | 1 | 2 | 1 | 2 | 2 | 5 | 33 | 0 | 2 | 1 |
| Jazz | 1 | 0 | 3 | 1 | 0 | 5 | 28 | 0 | 0 | 0 |
| Latin | 2 | 1 | 1 | 4 | 4 | 20 | 52 | 0 | 2 | 0 |
| Pop_Rock | 5 | 5 | 11 | 9 | 0 | 14 | 580 | 5 | 2 | 1 |
| Rap | 0 | 0 | 2 | 2 | 0 | 3 | 36 | 35 | 1 | 0 |
| Reggae | 1 | 0 | 4 | 2 | 0 | 5 | 13 | 4 | 11 | 0 |
| RnB | 1 | 2 | 2 | 2 | 0 | 3 | 30 | 2 | 2 | 0 |

Table 5.7: Confusion matrix of the predictions of the test set obtained by kNN when using MKPOE-LR (for the full medium-scale dataset). The row represents the true class and the column represents the predicted class.

## 5.7 Conclusion

Multiple approaches to reduce the execution time of MKPOE are explored in this chapter. OMDL, an online adaptation of MKPOE, has a high execution time, thus failing to reduce the time of MKPOE. Some approaches result in poor performance of the metric learned, this is the case for MKPOE with diagonal matrices and for the Frank-Wolfe formulation. And other approaches allow to learn a relatively performing metric while significantly reducing the training time. For example, MKPOE-LR learns low-dimensional parameter matrices thanks to random projections, and the mini-batch stochastic gradient descent decreases the time of the iteratin algorithm by selecting a subset of the distance constraints at each iteration.

# Conclusion

Metric learning is often needed to learn a task-specific metric. Meanwhile, multi-modal data increasingly arise in real-world cases, such as multimedia applications. Metric learning techniques that can efficiently deal with these multiple modalities have a growing importance. This master's thesis shows that taking advantage of the multi-modal aspect of the data allows to increase the prediction accuracy of a musical genre recognition task. The modalities are combined following the method proposed in MKPOE, a state-of-the-art technique for multi-modal metric learning. But MKPOE has limitations on some datasets. The experiments show its inability to learn a metric that generalizes to unseen data when the number of training examples per class is small, such as in an artist recognition task.

In addition, MKPOE does not scale well with the size of the training set. Therefore, multiple approaches to reduce the execution time of this algorithm are explored. Among these, some are successful and well performing on the genre recognition task while having a significantly smaller execution time than MKPOE. First, the number of distance constraints can be limited by selecting a fixed number of target neighbors and impostors for each training point. The accuracy of MKPOE when the number of impostors is fixed is comparable to the accuracy obtained when selecting all the points from a different class as impostors. Another approach that performs well is to use a mini-batch subgradient descent, where a subset of the constraints is selected at each iteration. This master's thesis shows that the "bold driver" technique is convenient to easily update the learning rate of the subgradient descent algorithm. But it is not possible to efficiently use this technique with the mini-batch subgradient descent, because, in order to compute the cost of the objective function at each iteration, the "bold driver" technique needs to pass over all the distance constraints. Finally, in order to reduce the time needed to project onto the set of positive semi-definite matrices, the MKPOE-LR algorithm is proposed. MKPOE-LR learns low-dimensional parameter matrices of the metric by using random projections. For some datasets, the accuracy obtained with this technique is comparable to the accuracy obtained with MKPOE.

## Further research

In order to validate the proposed techniques, such as MKPOE-LR, their performance should be assessed on other multi-modal datasets. Testing with multiple datasets could also be done to confirm whether the performance increase brought by these techniques is statistically significant or not. Indeed, despite an increase in the accuracy, the results of this master's thesis don't show a statistically significant difference between the accuracy obtained with the metric learning techniques and the accuracy obtained with a kNN classifier without a metric learning step. It would also be interesting to apply the techniques to other tasks involving multi-modal data. For example, with the Million Song Dataset, a metric learning step could improve the performance of a year recognition task (i.e. a regression task) or a content-based playlist generation.

One could explore other iterative algorithms in order to efficiently optimize the metric learning

problem. Some algorithms are known to have a faster convergence rate than a classical gradient descent. For instance, this is the case for the Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) [2].

To be complete, the performance should be compared with the multi-wing harmonium model proposed in [28], another state-of-the-art technique for multi-modal metric learning.

In order to avoid the time cost of projecting onto the set of PSD matrices, some metric learning techniques use a LogDet divergence as regularization term. This regularizer enforces the parameter matrix $M$ to be close to an initial value $M_0$ and ensures that $M$ is always PSD. One limitation of this approach is that the choice of $M_0$ (often set to $M_0 = I$) has an influence on the performance of the learned distance [4]. Examples of techniques that use this regularizer are Information-Theoretic Metric Learning (ITML) [10] and LogDet Exact Gradient Online (LEGO) [14]. As LEGO is an online metric learning approach, it would be interesting to study if OMDL can take advantage of the LogDet divergence in order to avoid the projection cost, by combining the ideas of OMDL and LEGO.

Exploring the use of linear metric learning techniques with multi-modal data is something that has rarely been done (see for example [26]). Some future work can compare the performance of linear metric learning techniques with the ones proposed in this master's thesis. The advantage of the multi-modal aspect of the data when using a linear metric learning technique is less obvious than with kernelized nonlinear techniques. In the latter, the choice of the kernel can be justified by prior knowledge of the modality. Also, if the modalities are simply concatenated, the kernel entries are computed on all the features, but the learned parameter matrix is applied on the kernel entries. Whereas in linear metric learning techniques it is applied directly on the features. But learning a linear metric has multiple advantages. For example there is no need to choose the kernels and to tune their hyperparameters, and it reduces the probability of overfitting because a simpler model is learned. If the data is very high-dimensional ($d \gg n$), then more parameters must be learned because the size of the parameter matrix $M$ is $d \times d$ for linear metric learning techniques, instead of $n \times n$ for kernelized nonlinear techniques.

Finally, a possible approach to scale the metric learning algorithms to large datasets could be to select a subset of the data points and learn the metric based only on this subset. In comparison, the mini-batch subgradient descent selects a subset of the distance constraints at each iteration. Mini-batch is studied in this master's thesis, but selecting a subset of the training points is not explored. Selecting promising data points that represent well the underlying geometry of the data can be done with active learning. Further research can study the performance of MKPOE or MKPOE-LR when using this approach.

# Bibliography


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

[2] Amir Beck, and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

[3] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. arXiv preprint arXiv:1306.6709, 2013.

[4] Aurélien Bellet, Amaury Habrard, and Marc Sebban. Metric Learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan & Claypool Publishers, 2015.

[5] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR)*, 2011.

[6] Ella Bingham, and Heikki Mannila. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 245–250, 2001.

[7] Gregory W. Corder, and Dale I. Foreman. Nonparametric Statistics: A Step-by-Step Approach, 2nd Edition. *John Wiley & Sons*, 2014.

[8] Thomas Cover, and Peter Hart. Nearest neighbor pattern classification. In *IEEE Transactions in Information Theory 13*, pages 21–27, 1967.

[9] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. Online Passive-Aggressive Algorithms. In *Journal of Machine Learning Research*, pages 551–585, 2006.

[10] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216, 2007.

[11] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. In *Journal of Machine learning research 7*, pages 1–30, 2006.

[12] Marguerite Frank, and Philip Wolfe. An algorithm for quadratic programming. *Naval research logistics quarterly*, 3(1-2):95–110, 1956.

[13] Martin Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[14] Prateek Jain, Brian Kulis, Inderjit S. Dhillon, and Kristen Grauman. Online metric learning and fast similarity search. In *Advances in neural information processing systems (NIPS)*, pages 761–768, 2009.

[15] Rong Jin, Steven C.H. Hoi, and Tianbao Yang. Online Multiple Kernel Learning: Algorithms and Mistake Bounds. In *Algorithmic Learning Theory: 21st International Conference (ALT)*, pages 390–404, 2010.

[16] William B. Johnson, and Joram Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Contemporary mathematics 26*, pages 189–206, 1984.

[17] Brian Kulis. Metric Learning: A Survey. In *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.

[18] Kuan Liu, Aurélien Bellet, and Fei Sha. Similarity Learning for High-Dimensional Sparse Data. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[19] Brian McFee, and Gert Lanckriet. Learning Multi-modal Similarity. In *Journal of machine learning research 12*, pages 491–523, 2011.

[20] Qi Qian, Rong Jin, Jinfeng Yi, Lijun Zhang, and Shenghuo Zhu. Efficient distance metric learning by adaptive sampling and mini-batch stochastic gradient descent (SGD). *Machine Learning 99.3*, pages 353–372, 2015.

[21] Matthew Schultz, and Thorsten Joachims. Learning a distance metric from relative comparisons. In *Advances in neural information processing systems (NIPS) 16*, 2004.

[22] Chunhua Shen, Junae Kim, Lei Wang, and Anton van den Hengel. Positive semidefinite metric learning with boosting. In *Advances in neural information processing systems (NIPS)*, pages 1651–1659, 2009.

[23] Yuan Shi, Aurélien Bellet, and Fei Sha. Sparse Compositional Metric Learning. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2078–2084, 2014.

[24] Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1473–1480, 2006.

[25] Kilian Q. Weinberger, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Journal of Machine Learning Reasearch (JMLR) 10*, pages 207–244, 2009.

[26] Pengcheng Wu, Steven C.H. Hoi, Peilin Zhao, Chunyan Miao, and Zhi-Yong Liu. Online Multi-modal Distance Metric Learning with Application to Image Retrieval. In *IEEE Transactions on Knowledge and Data Engineering*, 28(2):454–467, 2016.

[27] Hao Xia, Pengcheng Wu, and Steven C.H. Hoi. Online Multi-modal Distance Learning for Scalable Multimedia Retrieval. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 455–464, 2013.

[28] Pengtao Xie, and Eric P. Xing. Multi-modal Distance Metric Learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**