Imperial College London

Department of Computing

Graph regularized Semi- and Convex-Nonnegative Matrix factorization for clustering Static and Dynamic Data

by
Panagiotis Tzirakis (pt511)

Submitted in partial fulfilment of the requirements for the MSc Degree in Advanced Computing of Imperial College London

September 6, 2013

Abstract

Nonnegative Matrix Factorization (NMF) is a very famous technique that can be used for clustering or dimensionality reduction. In this thesis, we used NMF as clustering technique. In addition, we also dealt with two variations of NMF. These are the Semi-NMF and Convex-NMF. These techniques extend the applicability of NMF in that they can deal with negative values. Moreover, a generalization of the NMF has been developed which takes into account the intrinsic geometry of the data. This generalization is called Generalized Nonnegative Matrix Factorization (GNMF).

In this project, we propose two new methods. These methods are the Graph Semi-NMF and Graph Convex-NMF, which are generalizations of the Semi-NMF and Convex-NMF, respectively. Like GNMF, these methods take into account the geometry of the data. Moreover, we use Convex-NMF and Graph Convex-NMF with two time series algorithms in order to enhance their performance. Finally, we present a comparison of all the algorithms and we conclude that our algorithms outperform the existing algorithms, in most datasets.

Acknowledgements

I want to thank my supervisor Dr. Stefanos Zafeiriou for his guidance throughout this project. Sincerest gratitude is expressed towards my family for their support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce some basic knowledge on clustering. First, we define the notion of clustering and its categories. Then, we present some popular similarity measures that are used in clustering. In addition, well known algorithms for clustering are discussed. Moreover, we present how non-negative matrix factorization can be used for clustering. Finally, we show how clustering can be applied in time series.

1.1 Clustering

Clustering is an unsupervised learning technique, which tries to find hidden structure in unlabelled data. The goal of clustering is to group (cluster) the data points in such a way that the data points that are in the same group are similar and the data points that are in different groups are dissimilar.

1.2 Categories of Clustering

According to Han and Kamber[1] there are five major categories in clustering:

1. Partitioning methods
2. Hierarchical methods
3. Density-based methods
4. Grid-based methods
5. Model-based methods

1.2.1 Partitioning methods

Partition methods divide the datasets into k clusters (k is predefined) such that each data point is in only one cluster. More formally,

Definition 1. *Given a dataset $X \in \Re^{d \times n}$ and a number $k \leq n$ (usually $k << n$), a partition method clusters each data point in X into one of the k clusters.*

Examples of algorithms that belong in this category is k-means[2], where the cluster is represented by the mean value of the data points belonging to the cluster, k-medoids[3], where the clusters are represented by the most centrally located objects in the cluster.

The assignment of each data point, for the predefined algorithms, is hard. That is, they belong to only one clsuter. There are also the algorithms fuzzy c-means[4] and fuzzy c-medoid[5]. These algorithms belong to fuzzy partitions, in which the assignment of data points to clusters in not hard. That is, each data point belongs to a cluster with a probability. For example, suppose that we have a data point x and 2 clusters. Then x may belong to cluster 1 with probability 40% and to cluster 2 with probability 60%.

1.2.2 Hierarchical methods

Hierarchical methods construct a hierarchy of clusters by recursively partitioning the instances. This can be done in either top-down or bottom-up. There are two main types of hierarchical clustering:

1. Agglomerative approaches, in which initially all objects belong to one cluster (its own) and then the clusters are successively merged into larger and larger clusters until all objects are in one single cluster or until certain termination conditions, such as the desired number of clusters, are satisfied.

2. Divisive methods do the opposite. In the beginning all objects belong to one (same) cluster and then the cluster is divided into smaller sub-clusters and these into even smaller sub-clusters until the desired cluster structure is obtained.

The merging or division of clusters is performed according to a similarity measure.

1.2.3 Grid-based methods

The Grid-Based methods "gridify" the data space. That is, a specified number of cells are created that form a grid structure. Then, clustering is performed on that space. This method is computationally efficient with low operation time. Well known algorithms of this method are the STING[8] and CLIQUE[9].

1.2.4 Density-based methods

Density-based methods assume that there is a probability distribution in which the points of each cluster are drawn from. The distribution of each cluster can be different from other clusters in the data.

The key idea behind these methods is to continue growing a cluster until the density in a neighbourhood exceeds some threshold. Popular algorithms are the DBSCAN[6] and OPTICS[7].

1.2.5 Model-based methods

These methods try to find mathematical models to fit the data. The idea is to find some object characteristics within the cluster. Decision trees and neural networks are some very well known model-based methods.

1.3 Similarity measures

In clustering it is very important to define similarities measures. These similarities, measure how alike are two data points. Here we describe the most well known similarities measures.

1.3.1 Euclidean distance, Root Mean Square distance and Mikowski distance

Euclidean distance is a very popular similarity measure and it is defined as follows. Given two vectors x, y the Euclidean distance is given by

$$d_E(x, y) = \sqrt{\|x - y\|^2}$$

The Root Mean Square (RMS or average geometric distance) is given by

$$d_RMS(x, y) = d_E(x, y)/N$$

where N is the length of the vectors. Finally, the Mikowski distance, which is a generalization of the Euclidean distance, is given by

$$d_M(x, y) = \sqrt[q]{\|x - y\|^q}$$

1.3.2 Pearson's correlation

Another popular similarity measure is the Pearsons correlation factor. Pearson's correlation measures the linear dependence of two variables and can take values $[-1, 1]$, with -1 to denote negative correlation, +1 positive correlation and 0 uncorrelated. This measure is defined as follows

$$cc = \frac{E[(x - \mu_x)(y - \mu_y)]}{S_x S_y}$$

where μ_x and S_x is the mean of the vector and the scatter matrix, respectively. The scatter matrix is calculated as follows

$$S_x = \sqrt{x - \mu_x}$$

1.3.3 Short time series distance

Moller-Levet et al. [11] proposed the Short Time Series (STS) distance between two time series x and y as follows

$$d_{STS} = \sqrt{\sum_{k=1}^{p} (\frac{x_{j(k+1)} - x_{jk}}{t_{(k+1)} - t_k} - \frac{y_{j(k+1)} - y_{jk}}{t_{(k+1)} - t_k})^2}$$

where t_k is the time point for the data point x and y.

1.3.4 Dynamic Time Warping(DTW) distance

Dynamic Time Warping (DTW) distance is used to align two or more time-dependent sequences. It can be used to align discrete sequences with sequences of continuous values. These sequences can have different length.

Consider two time series $X = x_1, \ldots, x_n$ and $Y = y_1, \ldots, y_m$. DTW aligns these sequences so that their difference is minimized. For this purpose a $(n \times m)$ distance matrix is created where the (i, j) element contains the distance $d(x_i, y_j)$ (normally the Euclidean distance is used) between two points x_i and y_j. After having found the distance matrix the warping path $W = w_1, w_2, \ldots, w_K$ is found where $max(n, m) \leq K \leq m + n - 1$. This path must satisfy the three following conditions:

1. *Boundary condition*: The first and the last point of the warping path must be the first and the last points of the aligned sequence, $w_1 = (1, 1)$ and $w_K = (N, M)$.

2. *Monotonicity condition*: Preserves the time-ordering of points.

3. *Continuity condition*: Two successive points in the warping path must be in neighbour cells.

It is computationally inefficient to find the warping path by testing every possible warping path between X and Y. That is why dynamic programming is used.

1.3.5 Kullback Leibler distance

The Kullback-Leibler distance is defined over two probabilities distributions P and Q on a finite set S as follows

$$KL(P\|Q) = \sum_{i=1}^{S} P \log \frac{P}{Q}$$

1.4 Well known Clustering Algorithms

In this subsection we describe some basic algorithms that have extensively been used. Some of the algorithms that we describe in later chapters are extensions of these algorithms.

1.4.1 K-means, Kernel k-means and fuzzy c-means

K-means[2] is one of the simplest clustering algorithms which tries to divide the data into k (known a priori) clusters. Each observation belongs to only one cluster. This is the cluster with the nearest mean (hard assignment). The algorithm tries to minimize the following objective function

$$J_{K-Means} = \sum_{j=1}^{N} \sum_{i=1}^{K} g_{ij} \|x_i - c_i\|^2$$

where N is the number of data, K is the number of clusters, x_i is a data point, c_i is a cluster center and g_{ij} is defined as follows

$$g_{ij} = \begin{cases} 1, \text{if } x_j \text{ is assigned to cluster i} \\ 0, otherwise \end{cases}$$

To minimize the objective function the algorithm alternates between two steps

- Assignment step: Assign each data point to the closest cluster c_k.
- Refitting step: Each cluster centre is moved to the centre of gravity of the data assigned to it. That is

  $$c_j = \frac{1}{N_k} \sum_{i=1}^{N_k} x_i$$

  where N_k is the number of points that belong to cluster i.

The convergence of K-means is guaranteed because whenever an assignment is changed or whenever a cluster centre is moved the sum of squared distances of the data points from their currently assigned cluster centres is reduced.

A variation of k-means is fuzzy c-means[4] where each data point belongs to a cluster centre with a probability (soft assignment). Thus, a point between two clusters may belong to one cluster with more probability than the other cluster. The objective function of this algorithm is the following:

$$J_{FuzzyC-Means} = \sum_{j=1}^{N} \sum_{i=1}^{K} g_{ij}^m \|x_i - c_j\|^2$$

where m is a real number greater than 1 and

$$c_j = \frac{\sum_{i=1}^{N} g_{ij}^m x_i}{\sum_{i=1}^{N} g_{ij}^m}$$

$$g_{ij} = \frac{1}{\sum_{k=1}^{K} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|}\right)^{\frac{2}{m-1}}}$$

The optimization procedure to solve the fuzzy c-means is the following:

1. Choose c cluster centres, select m and ε (very small number). ε is used in termination condition. Compute the membership matrix $G^{(0)}$, where the element (i, j) of the matrix has the value g_{ij}.
2. Calculate the cluster centres c_i.
3. Compute an updated membership matrix $G^{(k+1)}$ by using the equation of g_{ij}.

4. Compare $G^{(k+1)}$ to $G^{(k)}$. If $G^{(k+1)} - G^{(k)} < \varepsilon$, then stop. Otherwise, set $G^{(k)} = G^{(k+1)}$ and go to step 2.

One disadvantage of k-means and fuzzy c-means is that they cannot separate clusters that are non-linearly separable in input space. To overcome this problem Kernel K-Means[12] was introduced. This algorithm uses a non-linear mapping from the input space to a higher dimensional space and performs the conventional k-means in that space. The objective function of the Kernel K-Means is the following:

$$J_{KernelK-Means} = \sum_{i=1}^{N}\sum_{j=1}^{K} g_{ij}\|\phi(x_i) - c_j\|^2$$

where

$$c_i = \frac{\sum_{j=1}^{N} g_{ij}\phi(x_j)}{\sum_{j=1}^{N} g_{ij}}$$

The distance between the cluster and the data point would be

$$\begin{aligned}\|\phi(x_i) - c_k\|^2 &= \phi(x_i)\phi(x_i) - 2\frac{\sum_{j=1}^{N}\phi(x_i)\phi(x_j)}{\sum_{j=1}^{N} g_{kj}} + \frac{\sum_{j,z}^{N}\phi(x_j)\phi(x_z)}{\sum_{j=1}^{N}\sum_{z=1}^{N} g_{kj}g_{kz}} \\ &= K(x_i, x_i) - 2\frac{\sum_{j=1}^{N} K(x_i, x_j)}{\sum_{j=1}^{N} g_{kj}} + \frac{\sum_{j,z}^{N} K(x_j, x_z)}{\sum_{j=1}^{N}\sum_{z=1}^{N} g_{kj}g_{kz}}\end{aligned}$$

where $K(\cdot)$ denotes the kernel matrix.

1.4.2 Agglomerative hierarchical clustering

As described earlier there are two types of hierarchical clustering: agglomerative and divisive. Here we describe the agglomerative approach as it is the most famous one.

In this type of hierarchical clustering at the beginning of the algorithm, each object is considered to belong to one cluster (its own). The next step is to measure the similarity of each cluster with every other cluster and then merge the most similar together. This procedure takes place until one cluster is created or when a termination criterion is satisfied. Depending on the measure of similarity between clusters three main types can be considered:

Single-link Clustering

In these methods the distance between any two clusters is considered to be the shortest distance between these clusters. That is, the shortest distance of a data point that belongs to the first cluster and a data point that belongs to the second cluster.

Complete-link Clustering

Unlike the Single-Link methods in Complete-link methods the distance between two clusters is the maximum distance between these clusters. That is, the maximum distance between a data point that belongs to one cluster and a data point that belongs to a second cluster.

Average-link Clustering

In these methods the distance between two clusters is considered to be the average distance between these clusters. That is, the average distance between all the points of both clusters.

1.4.3 Self-Organizing Map

Self-Organizing Map (SOM) was developed by Kohonen [13] and it is a special case of neural network. This method converts high-dimensional input to low-dimensional and it is trained by an iterative self-organizing procedure.

The learning process of the algorithm consists of the following steps:

1. Initialize each node with small random values of weights.
2. Choose a random vector from the input space and present it to the network.
3. Find the Best Matching Unit (BMU) in the network by calculating the distance between the weight vector of each node and the input vector.
4. Determine the BMU neighbourhood. The neighbourhood at each iteration shrinks.
5. Alter the weights for the nodes in the neighbourhood. The closest to the BMU the more change in the weights of a node.
6. Repeat from step 2 until convergence.

1.5 Nonnegative Matrix Factorization for Clustering

Nonnegative Matrix Factorization (NMF) can approximately decompose a non-negative matrix X into two non-negative lower-rank matrices F and G. This technique can be used for clustering in which the matrix X is the data matrix, the matrix F is the centroid matrix, which represents where the cluster centre are in the space, and the matrix G is the cluster indicator matrix, which indicates in which cluster each data point belongs. Depending on the cost function that is used for the approximation, different algorithms can be created. Variations of NMF have been proposed. In this thesis we deal with two variations. These are the Semi-NMF and Convex-NMF, which extends the range of application of NMF, in that they can be used with mixed signed data. These techniques are described in the next chapter more thoroughly.

1.6 Time Series Clustering

In this project we also deal with time series datasets. So it is necessary to explain how clustering can be made in such datasets. As in conventional clustering, in time series datasets we try to group unlabelled data objects. The type of algorithm that can be used depends on the data, which can be multivariate or univariate, discrete-value or real-value etc. and the application. According to [10] time series can be distinguished into three types (Figure 1.6.1):

Raw-data-based
In this approach clustering can be applied directly to raw time series data (thus called raw-data-based). The similarity measure used in this case is one appropriate for time series.

Feature-based
In this approach the raw time series data are first converted into a feature vector of lower dimension and then conventional clustering is applied.

Modeled-based
This approach first converts the raw time series data into a number of model parameters and then either the model parameters can be used directly without applying a conventional clustering algorithm or a clustering algorithm can be used.

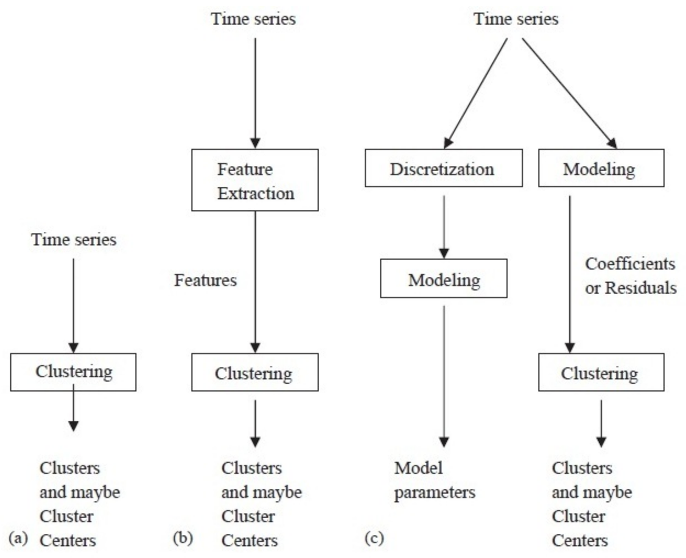

Figure 1.6.1: Types of Time Series Clustering

Chapter 2

Graph regularized Semi- and Convex- Nonnegative matrix Factorization

This chapter is divided in two parts. In the first part, we describe the Nonnegative Matrix Factorization(NMF) and its variation Semi- and Convex-NMF. In the second part of, we describe the Graph regularized algorithms. That is, the Graph regularized Nonnegative Matrix Factorization (GNMF) and the variations that we propose the Graph regularized Semi- and Convex-NMF.

2.1 Nonnegative Matrix Factorization (NMF)

This section describes the Nonnegative Matrix Factorization (NMF) technique and presents some algorithms that has been used in this technique. Later, we mention two extension of the NMF. These are the Semi- and Convex-NMF. Again we present algorithms that are used for these techniques. Moreover, we describe two variations of Convex-NMF, the Kernel- and Cluster-NMF. Finally, an example is given for Semi- and Convex-NMF.

2.1.1 Introduction

Non-negative Matrix Factorization (NMF) was first introduced by P. Paatero and U. Tapper [14], and finds its origin in numerical linear algebra. NMF tries to factorize a matrix into two non-negative matrices. The idea behind the non-negativity of the matrices lies on the fact that in many datasets negative numbers have no physical meaning. Applications of NMF can be found in text mining[15][16], bioinformatics[17][18] and sound recognition[19][20].

NMF can be used either as a clustering or dimensionality reduction technique. For this project NMF has been used as a clustering technique. Formally, the NMF problem can be defined as followed:

Definition 2. *Given a non-negative data matrix $X \in \Re_+^{d \times n}$ find two non-negative lower rank data matrices $F \in \Re_+^{d \times m}$ and $G \in \Re_+^{m \times n}$ with $m < (d, n)$ such that*

$$X \approx FG \tag{2.1}$$

The NMF decomposition can be solved in repeated iterations where first the matrix F is considered to be constant and G is updated and then G is considered fixed and F is updated. There are different cost functions that can be used in order to take the approximation in 2.1. In [21] they found two algorithms. They first considered the following cost functions:

1. The Euclidean distance between A and B:

$$\|A - B\|^2 = \sum_{ij} (A_{ij} - B_{ij})^2$$

2. The "divergence" as they refer it

$$D(A\|B) = \sum_{ij} (A_{ij} log \frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij})$$

For these cost functions they proved that the multiplicative update rules are the following

1. For the square of the Euclidean distance:

$$G_{\alpha\mu} \leftarrow G_{\alpha\mu} \frac{(F^T X)_{\alpha\mu}}{(F^T F G)_{\alpha\mu}}, F_{\iota\alpha} \leftarrow F_{\iota\alpha} \frac{(X G^T)_{\iota\alpha}}{(F G G^T)_{\iota\alpha}}$$

2. For the "divergence"

$$G_{\alpha\mu} \leftarrow G_{\alpha\mu} \frac{\sum_i F_{\iota\alpha} X_{\iota\mu}/(FG)_{\iota\mu}}{\sum_k F_{\kappa\alpha}}, F_{\iota\alpha} \leftarrow F_{\iota\alpha} \frac{\sum_\mu G_{\alpha\mu} X_{\iota\mu}/(FG)_{\iota\mu}}{\sum_v G_{\alpha v}},$$

Many other algorithms have been created. The algorithm used, depends on the problem that is being solved. For example, in [22] they found an algorithm that is able to learn parts of faces and semantic features of text. The update rules that they found are the following:

$$G_{\alpha\mu} \leftarrow G_{\iota\alpha} \sum_{\mu} \frac{X_{i\mu}}{(WH)_{i\mu}} F_{\alpha\mu}, F_{\alpha\mu} \leftarrow F_{\alpha\mu} \sum_{i} W_{i\alpha} \frac{X_{i\mu}}{(WH)_{i\mu}},$$

As mentioned earlier, NMF can be used as a clustering technique where the matrix F contains the clusters position and G contains the cluster where each data point belongs. For example, K-means objective function can be written in a form using NMF. That is,

$$J_{k-means} = \sum_{i=1}^{n} \sum_{k=1}^{K} g_{ik} \|x_i - f_k\|^2 = \|X_+ - F_+ G_+\|_F^2$$

where $g_{ik} = 1$ if data point i belongs to k cluster;$g_{ik} = 0$ otherwise. f_k is the cendroid of the cluster k.

2.1.2 Semi - NMF

An extension of NMF was introduced in [23]. This is the Semi-NMF which can also be used in clustering. The decomposition is of the following form:

$$X \approx FG^T$$

where$X \in \Re_+^{d\times n}$ is the data matrix, $F \in \Re_\pm^{d\times m}$ contains the cluster centroids and $G \in \Re_+^{n\times m}$ is the cluster indicator matrix. In their paper they suggested to relax G so that its values would range between $(0, \infty)$ and F could also take negative values. The main difference between the Semi-NMF and NMF is that the matrix centroid F can take positive and negative values.

The algorithm they suggested is the following:

Initialization

Initialization of G can be performed in two ways:

1. K-means initialization: Run K-means clustering algorithm to find the cluster indicator matrix G and then add a small value (they suggest 0.2) to all elements of G.
2. Random initialization: Randomly initialize G matrix with values ranging between $(0, \infty)$.

Iterative rules

Iterate between the following update rules until convergence

- Update the centroid matrix F while keeping G fixed.

  $$F = XG(G^TG)^{-1}$$

  The computational complexity is of order $p(dnm + nm^2)$, where p is the number of iterations.

- Update the cluster indicator matrix G while keeping F fixed.

  $$G_{ik} \leftarrow G_{ik}\sqrt{\frac{(X^TF)^+_{ik} + [G(F^TF)^-]_{ik}}{(X^TF)^-_{ik} + [G(F^TF)^+]_{ik}}}$$

  where

  $$A^+_{ik} = (|A_{ik}| + A_{ik})/2$$
  $$A^-_{ik} = (|A_{ik}| - A_{ik})/2$$

  The computational complexity is of order $p(ndm + md^2 + n^2m)$, where p is the number of iterations.

2.1.3 Convex - NMF

In [23] they proposed one more variation of NMF in which the centroid matrix F is constrained to be a convex combination of the column of X. That is F lies in the columns space of X:

$$f_l = w_{1l}x_1 + \cdots + w_{nl}x_n = Xw_l, \text{ or } F = XW$$

where $W \in \Re^{n\times m}_+$. So the Convex-NMF form will be the following:

$$X_\pm \approx X_\pm W_+ G^T_+$$

The algorithm that they proposed is the following:

Initialization

Initialization of W and G can be performed in two ways:

1. K-means initialization: Run K-means clustering algorithm to find the cluster indicator matrix G and then add a small value (they suggest 0.2) to all elements of G. For example, if the result of k-means is the cluster indicator matrix $H = (h_1, \ldots, h_k)$ then $G^{(0)} = H + 0.2 \cdot 1_k$.

   Then, the cluster centroid can be computed as $f_k = Xh_k/n_k$ or $F = XHD_n^{-1}$, where $D_n = diag(n_1, \ldots, n_k)$. So $W = HD_n^{-1}$

2. Have NMF or Semi-NMF solution: If there is a solution of NMF or Semi-NMF then G can be computed as $G^{(0)} = G + 0.2 \cdot 1_k$ and $W = G(G^TG)^{-1}$. Because W should be nonnegative, then $W^{(0)} = W^+ + 0.2 \cdot 1_k \langle W^+ \rangle$ where $\langle A \rangle = \sum_{ij} |A_{ij}| / \|A\|_0$.

Iterative rules

The algorithm iterates between the following update rules until convergence

- Update the cluster indicator matrix G while keeping W fixed.

$$G_{ik} \leftarrow G_{ik} \sqrt{\frac{[(X^TX)^+W]_{ik} + [GW^T(X^TX)^-W]_{ik}}{[(X^TX)^-W]_{ik} + [GW^T(X^TX)^+W]_{ik}}}$$

  The computational complexity is of order $n^2d + p(2n^2m + nm^2)$, where p is the number of iterations.

- Update the W matrix while keeping G fixed.

$$W_{ik} \leftarrow W_{ik} \sqrt{\frac{[(X^TX)^+G]_{ik} + [(X^TX)^-WG^TG]_{ik}}{[(X^TX)^-G]_{ik} + [(X^TX)^+WG^TG]_{ik}}}$$

  The computational complexity is of order $p(2n^2m + 2nm^2)$, where p is the number of iterations.

Kernel NMF

The Convex-NMF algorithm can be used with kernels, which it is not possible if NMF or Semi-NMF is used. Suppose a mapping function such that $X \rightarrow \phi(X) = (\phi(x_1), \ldots, \phi(x_n))$. Then, using the Convex-NMF approximation for X with that mapping function we will have the following:

$$\phi(X) \approx \phi(X)WG^T$$

So the objective function will be:

$$\begin{aligned} \|\phi(X) - \phi(X)WG^T\|^2 = Tr[\phi(X)^T\phi(X)] - 2Tr[G^T\phi^T(X)\phi(X)W] \\ +Tr[W^T\phi^T(X)\phi(X)WG^TG] \end{aligned} \tag{2.2}$$

If we consider a kernel matrix $K = \phi^T(X)\phi(X)$ then (2.2) can be written in the following form

$$\|\phi(X) - \phi(X)WG^T\|^2 = Tr[K] - 2Tr[G^TKW] + Tr[W^TKWG^TG]$$

The update rules of the Kernel-NMF is similar to Convex-NMF with the difference that the term (X^TX) is replaced with $\phi^T(X)\phi(X) = K$. So the update rules will be the following

$$G_{ik} \leftarrow G_{ik}\sqrt{\frac{[(K)^+W]_{ik} + [GW^T(K)^-W]_{ik}}{[(K)^-W]_{ik} + [GW^T(K)^+W]_{ik}}}$$

$$W_{ik} \leftarrow W_{ik}\sqrt{\frac{[(K)^+G]_{ik} + [(K)^-WG^TG]_{ik}}{[(K)^-G]_{ik} + [(K)^+WG^TG]_{ik}}}$$

Cluster NMF

If we consider the elements of G to be posterior probabilities then it is proven that the cluster centroid matrix would be $F = XG$. So the approximation of X will be:

$$X \approx XG_+G_+^T$$

Thus the objective function will have the following form

$$\begin{aligned} J &= \|X - XGG^T\|^2 \\ &= Tr(X^TX) - 2Tr(GG^TX^TX) + Tr(GG^TX^TXGG^T) \end{aligned}$$

From the above equation we can conclude that the degrees of freedom depend only on G. That is why this NMF variation is called Cluster-NMF. The update rule for this algorithm will be

G update rule

G can take only positive values. So now the derivative for G would be.

$$\frac{dJ}{dG} = -2X^TXG + 4X^TXGG^TG = 0 \implies$$

$$-[(X^TX)^+ - (X^TX)^-]G + 2[(X^TX)^+ - (X^TX)^-]GG^TG = 0 \implies$$

$$[(X^TX)^-G + 2(X^TX)^+GG^TG] - [(X^TX)^+G + 2(X^TX)^-GG^TG] = 0 \implies$$

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(X^TX)^+G]_{ij} + 2[(X^TX)^-GG^TG]_{ij}}{[(X^TX)^-G]_{ij} + 2[(X^TX)^+GG^TG]]_{ij}}}$$

Cluster-NMF with kernel

In the previous section, we described the Cluster-NMF where we find an update rule for the cluster indicator matrix G. It is easy to see that we can use kernel with Cluster-NMF. Consider a mapping $X \rightarrow \phi(X)$. Now the objective function of the Cluster-NMF will have the following form.

$$\begin{aligned} J &= \|\phi(X) - \phi(X)GG^T\|^2 \\ &= Tr(\phi^T(X)\phi(X)) - 2Tr(GG^T\phi^T(X)\phi(X)) + Tr(GG^T\phi^T(X)\phi(X)GG^T) \\ &= Tr(K) - 2Tr(GG^TK) + Tr(GG^TKGG^T) \end{aligned}$$

where we substitute $\phi^T(X)\phi(X) = K$.

Doing the same calculation as in Cluster-NMF we will have the following update rule for the cluster indicator matrix G.

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(\phi^T(X)\phi(X))^+G]_{ij} + 2[(\phi^T(X)\phi(X))^-GG^TG]_{ij}}{[(\phi^T(X)\phi(X))^-G]_{ij} + 2[(\phi^T(X)\phi(X))^+GG^TG]]_{ij}}}$$

and if we substitute $\phi^T(X)\phi(X) = K$ we will have

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(K)^+G]_{ij} + 2[(K)^-GG^TG]_{ij}}{[(K)^-G]_{ij} + 2[(K)^+GG^TG]]_{ij}}}$$

2.1.4 Example

In this section we illustrate an example of Semi-NMF and Convex-NMF so as to make it more clear. Suppose we have the following data matrix

$$X = \begin{pmatrix} 1.5877 & -1.1480 & -1.9330 & 0.3035 & -0.8396 & -0.1977 \\ -0.8045 & 0.1049 & -0.4390 & -0.6003 & 1.3546 & -1.2078 \\ 0.6966 & 0.7223 & -1.7947 & 0.4900 & -1.0722 & 2.9080 \\ 0.8351 & 2.5855 & 0.8404 & 0.7394 & 0.9610 & 0.8252 \\ -0.2437 & -0.6669 & -0.8880 & 1.7119 & 0.1240 & 1.3790 \\ 0.2157 & 0.1873 & 0.1001 & -0.1941 & 1.4367 & -1.0582 \\ -1.1658 & -0.0825 & -0.5445 & -2.1384 & -1.9609 & -0.4686 \end{pmatrix}$$

where the data points are the columns and the attributes are the rows of X.

Then the cluster indicator matrix G would be

$$G_{semi} = \begin{pmatrix} 0.2546 & 1.0684 & 0.9440 & 0.7627 & 1.3635 & 0.0877 \\ 0.6591 & 0.8550 & 0.0201 & 1.1696 & 0.5157 & 1.2692 \end{pmatrix}$$

$$G_{convex} = \begin{pmatrix} 0.0147 & 0.1776 & 0.2167 & 0.1083 & 0.2510 & 0.0000 \\ 0.1242 & 0.1223 & 0.0000 & 0.2029 & 0.0103 & 0.2942 \end{pmatrix}$$

where for each data point the cluster indicator matrix indicates in which cluster the point belongs to. For exaple, for the first column in the G_{semi} matrix the point belongs to second cluster.

And the cluster centroid matrix F would be

$$F_{semi} = \begin{pmatrix} -1.2265 & 0.7117 \\ 0.7390 & -0.9206 \\ -1.6672 & 2.1382 \\ 0.8751 & 0.6189 \\ -0.7135 & 1.1495 \\ 0.8701 & -0.6727 \\ -0.7574 & -0.6418 \end{pmatrix}$$

$$F_{convex} = \begin{pmatrix} 0.0030 & 1.2293 \\ 0.5208 & 0.0025 \\ 1.8652 & 0.0015 \\ 0.0001 & 0.4346 \\ 2.1185 & 0.0043 \\ 0.0013 & 2.6474 \end{pmatrix}$$

where each column indicates the position of the centroid cluster.

2.2 Graph Regularized Non-negative Matrix Factorization (GNMF)

In this section, we describe an extension of the Non-negative factorization technique in which the structure of the data is taken into account. We provide the update rules of this algorithm. Furthermore, we describe two new techniques we created. These are the Graph Semi-NMF and Graph Convex-NMF. For these techniques we formulate the minimization problem and we prove the update rules. Finally, we show how kernels can be used with the Graph Convex-NMF.

2.2.1 Introduction

One disadvantage of NMF is that it does not take into account the geometric structure in the data. In [24] they proposed a method that overcomes this problem. By taking into consideration the manifold assumption[25], they considered the function $f_k(x_i) = u_{ik}$ which maps the data point x_i on the axis u_k. In order to measure the smoothness of the function they used the following

$$\|f_k\|_M^2 = \int \|\nabla_M f_k\|^2 dP_X(x)$$

where $M \subset \Re^m$ is a compact submanifold.

This manifold is not known in real datasets, but it can be approximated. They showed that a discrete approximation of that manifold is

$$R_k = g_k^T L g_k$$

where $L = D - W$ and W is the edge weight matrix which is defined as followed:

$$W_{ij} = \begin{cases} 1, & \text{if } x_i \in N_p(x_j) \text{ or } x_j \in N_p(x_i) \\ 0, & \text{otherwise} \end{cases}$$

where $N_p(x_j)$ is the number p nearest neighbours of x_j and D is a diagonal matrix where $D_{ii} = \sum_j W_{ij}$. So the objective function of the GNMF has the following form:

$$J = \|X - FG^T\|_F^2 + \lambda Tr(G^T LG)$$

The update rules that they found are the following:

$$F_{ij} = F_{ij} \frac{[XG]_{ij}}{[FG^T G]_{ij}}, G_{ij} = G_{ij} \frac{[X^T F + \lambda WG]_{ij}}{[GF^T F + \lambda DG]_{ij}}$$

It is worth noting, that if we take $\lambda = 0$ then the GNMF algorithm becomes the NMF algorithm.

2.2.2 Graph Semi-NMF

Considering the previous chapter and the current chapter where we described the Semi-NMF and a graph regularization of the NMF we develop a graph based approach for the Semi-NMF. In this approach we considered the same discrete approximation of $\|f_k\|_M^2$. So the objective function of Semi - NMF would be the following:

$$\begin{aligned} J &= \|X - FG^T\|_F^2 + \lambda Tr(G^T LG) \\ &= Tr(XX^T) - 2Tr(X^T FG^T) + Tr(GF^T FG^T) + \lambda Tr(G^T LG) \end{aligned} \tag{2.3}$$

which is the same as in the GNMF. For this case, though, the matrix F can take positive and negative values. So the update rules would be also different. The update rules are:

F update rule

Taking the derivative of (2.3) we have

$$\frac{dJ}{dF} = -2XG + 2FG^TG = 0 \implies$$
$$F = XG(G^TG)^{-1}$$

G update rule

G can take only positive values. So now the derivative for G would be.

$$\frac{dJ}{dF} = -2X^TF + 2GF^TF + 2\lambda LG = 0 \implies$$
$$-[(X^TF)^+ - (X^TF)^-] + G[(F^TF)^+ - (F^TF)^-] + \lambda[L^+ - L^-]G = 0 \implies$$
$$[(X^TF)^- + G(F^TF)^+ + \lambda L^+G] - [(X^TF)^+ + G(F^TF)^- + \lambda L^-G] = 0 \implies$$
$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(X^TF)^+ + G(F^TF)^- + \lambda L^-G]}{[(X^TF)^- + G(F^TF)^+ + \lambda L^+G]}}$$

2.2.3 Graph Convex-NMF

As in Semi-NMF we can take into account the intrinsic geometry of the data using the Convex-NMF. Recall that in Convex-NMF the centroid cluster matrix F is of the form $F = XW$. So the objective function would be:

$$\begin{aligned} J &= \|X - FG^T\|^2 + \lambda Tr(G^TLG) \\ &= \|X - XWG^T\|^2 + \lambda Tr(G^TLG) \implies \\ J &= Tr(X^TX) - 2Tr(G^TX^TXW) + Tr(W^TX^TXWG^TG) + \lambda Tr(G^TLG) \end{aligned} \tag{2.4}$$

The update rules for this algorithm will be the following:

W update rule

W can take only positive values. Taking the derivative of (2.4) with respect to W we have

$$\frac{dJ}{dW} = -2X^TXG^T + 2X^TXWG^TG = 0 \implies$$
$$-[(X^TX)^+ - (X^TX)^-]G^T + [(X^TX)^+ - (X^TX)^-]WG^TG = 0 \implies$$
$$[(X^TX)^-G^T + (X^TX)^+WG^TG + \lambda L^+G] - [(X^TX)^+G^T + (X^TX)^-WG^TG] \implies$$
$$W_{ij} = W_{ij}\sqrt{\frac{[(X^TX)^+G^T]_{ij} + [(X^TX)^-WG^TG]_{ij}}{[(X^TX)^-G^T]_{ij} + [(X^TX)^+WG^TG]_{ij}}}$$

G update rule

G can take only positive values. So taking the derivative of (2.4) with respect to G we have

$$\frac{dJ}{dG} = -2X^TXW + 2W^TX^TXWG + 2\lambda LG = 0 \implies$$

$$-[(X^TX)^+ - (X^TX)^-]W + W^T[(X^TX)^+ - (X^TX)^-]WG + \lambda[L^+ - L^-]G = 0 \implies$$

$$[(X^TX)^-W + W^T(X^TX)^+WG + \lambda L^+G] -$$

$$[(X^TX)^+W + W^T(X^TX)^-WG + \lambda L^-G] = 0 \implies$$

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(X^TX)^+W]_{ij} + [W^T(X^TX)^-WG]_{ij} + [\lambda L^-G]_{ij}}{[(X^TX)^-W]_{ij} + [W^T(X^TX)^+WG]_{ij} + [\lambda L^+G]_{ij}}}$$

Graph Kernel NMF

As in Convex-NMF algorithm, the Graph Convex-NMF algorithm can be used with kernels. Suppose a mapping function such that $X \to \phi(X) = (\phi(x_1), \ldots, \phi(x_n))$. Then using Graph Convex-NMF approximation of X with that mapping function we will have the following:

$$\phi(X) \approx \phi(X)WG^T$$

So the objective function will be:

$$\|\phi(X) - \phi(X)WG^T\|^2 + \lambda Tr(G^TLG) =$$
$$Tr[\phi(X)^T\phi(X)] - 2Tr[G^T\phi^T(X)\phi(X)W] + Tr[W^T\phi^T(X)\phi(X)WG^TG] + \lambda Tr(G^TLG) \tag{2.5}$$

If we consider a kernel matrix $K = \phi^T(X)\phi(X)$ then (2.5) can be written in the following form

$$\|\phi(X) - \phi(X)WG^T\|^2 + \lambda Tr(G^TLG) =$$
$$Tr[K] - 2Tr[G^TKW] + Tr[W^TKWG^TG] + \lambda Tr(G^TLG)$$

The update rules of the Kernel-NMF is similar to Convex-NMF with the difference that the term (X^TX) is replaced with $\phi^T(X)\phi(X) = K$. So the update rules will be the following

$$W_{ij} = W_{ij}\sqrt{\frac{[(K)^+G^T]_{ij} + [(K)^-WG^TG]_{ij}}{[(K)^-G^T]_{ij} + [(K)^+WG^TG]_{ij}}}$$

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(K)^+W]_{ij} + [W^T(K)^-WG]_{ij} + [\lambda L^-G]_{ij}}{[(K)^-W]_{ij} + [W^T(K)^+WG]_{ij} + [\lambda L^+G]_{ij}}}$$

Graph Cluster NMF

Considering the same form as in the previous chapter we introduce the Graph Cluster-NMF where the centroid cluster matrix is $F = XG$. So the approximation of X will be:

$$X \approx XG_+G_+^T$$

Thus the objective function will have the following form

$$\begin{aligned}J &= \|X - XGG^T\|^2 + \lambda Tr(G^T LG)\\ &= Tr(X^T X) - 2Tr(GG^T X^T X) + Tr(GG^T X^T XGG^T) + \lambda Tr(G^T LG)\end{aligned}$$

The update rule for this algorithm will be

G update rule

G can take only positive values. So now the derivative for G would be.

$$\frac{dJ}{dG} = -2X^T XG + 4X^T XGG^T G + \lambda Tr(G^T LG) = 0 \implies$$

$$-[(X^T X)^+ - (X^T X)^-]G + 2[(X^T X)^+ - (X^T X)^-]GG^T G + \lambda[L^+ - L^-]G = 0 \implies$$

$$\begin{matrix}[(X^T X)^- G + 2X^T X)^+ GG^T G + \lambda L^+ G] - \\ [(X^T X)^+ G + 2X^T X)^- GG^T G + \lambda L^- G] = 0\end{matrix} \implies$$

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(X^T X)^+ G]_{ij} + 2[(X^T X)^- GG^T G]_{ij} + \lambda[L^+ G]_{ij}}{[(X^T X)^- G]_{ij} + 2[(X^T X)^+ GG^T G]]_{ij} + \lambda[L^- G]_{ij}}}$$

Graph Cluster-NMF with kernel

It is possible to use kernel in the Graph Cluster-NMF algorithm. If we take again the mapping $X \to \phi(X) = (\phi(x_1), \ldots, \phi(x_n))$ then the objective function will have the following form

$$\begin{aligned}J &= \|\phi(X) - \phi(X)GG^T\|^2 + \lambda Tr(G^T LG)\\ &= Tr(\phi^T(X)\phi(X)) - 2Tr(GG^T \phi^T(X)\phi(X)) + Tr(GG^T \phi^T(X)\phi(X)GG^T) + \lambda Tr(G^T LG)\end{aligned}$$

The update rule will be the same as in the Cluster-NMF with the difference that X will be substitute with the mapping function $\phi(X)$. So it will be

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(\phi^T(X)\phi(X))^+ G]_{ij} + 2[(\phi^T(X)\phi(X))^- GG^T G]_{ij} + \lambda[L^+ G]_{ij}}{[(\phi^T(X)\phi(X))^- G]_{ij} + 2[(\phi^T(X)\phi(X))^+ GG^T G]]_{ij} + \lambda[L^- G]_{ij}}}$$

where we can take $\phi^T(X)\phi(X) = K$ and so the update rule will take the following form.

$$G_{ij} \leftarrow G_{ij}\sqrt{\frac{[(K)^{+}G]_{ij} + 2[(K)^{-}GG^{T}G]_{ij} + \lambda[L^{+}G]_{ij}}{[(K)^{-}G]_{ij} + 2[(K)^{+}GG^{T}G]]_{ij} + \lambda[L^{-}G]_{ij}}}$$

Chapter 3

Time Series Clustering

In this chapter, we discuss about two unsupervised learning algorithms that are being used to segment time series. These algorithms were used to cluster human motion datasets. Firstly, in this chapter a short introduction of how time series are clustered is discussed. Then, we mention the aligned cluster analysis algorithm and we explain how this algorithm works by providing the minimization function along with the algorithm that solves it. Later in the chapter, we describe an extension of the aligned cluster analysis algorithm which is the hierarchical aligned cluster analysis. Finally, we try to improve these two algorithms by incorporating the cluster nmf and the graph cluster nmf which were described in previous chapters.

3.1 Introduction

ACA[26] and its extension HACA[27] are described which segment a human motion into actions. These are unsupervised learning algorithms and are an extension of Kernel K-Means, which was described in chapter 1. These algorithms were used to cluster human motion movements. That is, they try to decompose these movements into actions. For example, we have the motion in Figure 3.1.1 where the human walks,rotates, jumps, walks and runs. The algorithm clusters the motion into four clusters walk,rotate,jump,run.

3.2 Aligned Cluster Analysis (ACA)

ACA is a generalization of the kernel k-means and it was used to decompose the human motion into actions. More formally, given a human motion sequence $X \in \Re^{d\times n}$ with n frames, ACA decomposes X into m segments. Each segment $Y_i = X_{[s_i,s_i+1)}$ belongs to one of k clusters, where s_i is the

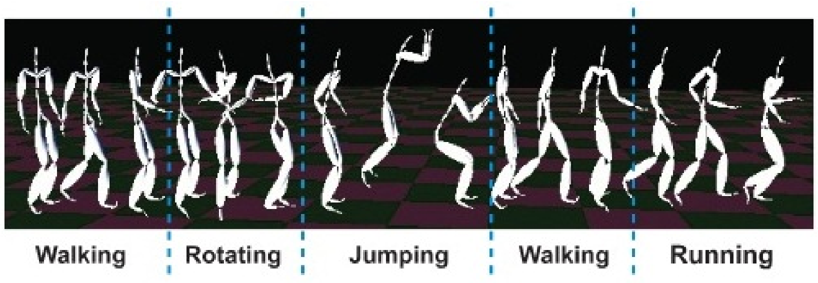

Figure 3.1.1: Human motion decomposed into actions. Source:[26]

position where the frames begin. The frames end in $s_{i+1} - 1$. Moreover, ACA constrains the maximum length of the segment to be n_{max}. That is, $s_{i+1} - s_i \leq n_{max}$.

3.2.1 Dynamic Time Alignment Kernel (DTAK)

As mentioned earlier ACA is a generalization of kernel k-means. The kernel that it uses is the dynamic time alignment kernel (DTAK), which measures the distance between segments. DTAK was proposed by [28] and it is an extension of the dynamic time warping (DTW) [29]. DTAK overcomes the disadvantage of DTW which was to satisfy the triangle inequality.

Given two sequences $X = [x_1, \dots, x_{n_x}] \in \Re^{d \times n_x}$ and $Y = [y_1, \dots, y_{n_y}] \in \Re^{d \times n_y}$ DTAK first computes a frame kernel matrix $K = \phi^T(X)\phi(Y) \in \Re^{n_x \times n_y}$. The elements of this matrix are calculated using a kernel function which basically defines the similarity between the frames. Examples of kernels are the Gaussian $\kappa_{ij} = \exp{(-\frac{\|x_i - y_j\|^2}{2\sigma^2})}$, the Laplacian $\kappa_{ij} = \exp{(-\frac{\|x_i - y_j\|}{2\sigma})}$ and the linear $\kappa_{ij} = x_i^T x_j$. After finding the frame kernel matrix DTAK finds the cumulative kernel matrix $U \in \Re^{n_x \times n_y}$ which its elements are computed as follows

$$u_{ij} = max \begin{cases} u_{i-1,j} + \kappa_{ij} \\ u_{i-1,j-1} + 2\kappa_{ij} \\ u_{i,j-1} + \kappa_{ij} \end{cases} \tag{3.1}$$

The initialization of the U matrix at the upper left is $u_{11} = 2\kappa_{11}$.

Now the similarity between the two sequences X, Y can be defined as follows

$$\tau(X, Y) = \frac{u_{n_x n_y}}{n_x + n_y} \tag{3.2}$$

DTAK can be also calculated using a matrix notation. First two frame indexes vectors are introduced. These vectors are $p \in \{1 : n_x\}^l$ and $q \in \{1 : n_y\}^l$, where l is the optimal length of the monotonic trajectory that can be found when aligning two sequences X,Y. The matrix W can be introduced and calculated as follows

$$W = [w_{ij}] \in \Re^{n_x \times n_y} \tag{3.3}$$

where

$$w_{ij} = \begin{cases} \frac{1}{n_x+n_y}(p_c - p_{c-1} + q_c + q_{c-1}) & \text{, if } p_c = i \text{ and } q_c = j \text{ for some c} \\ 0 & \text{, otherwise} \end{cases}$$

Using this matrix we can write DTAK in the form

$$\tau(X, Y) = tr(K^T W) = \psi(X)^T \psi(Y) \tag{3.4}$$

where $\psi(\cdot)$ declares the kernel function.

It is worth mentioning that DTAK is not always a strictly positive definite kernel [30][31] and by Mercer theorem [32] a mapping will not exist. This problem was solved and it is described later in the chapter.

3.2.2 Energy function

Considering all the above the energy function of ACA is the following

$$\begin{aligned} J_{aca}(G, s) &= \sum_{c=1}^{k} \sum_{i=1}^{m} g_{ci} \underbrace{\|\psi(X_{[s_i, s_{i+1})} - z_c)\|^2}_{dist_\psi^2(Y_i, z_c)} \\ &= \|[\psi(Y_1) \dots \psi(Y_m)] - ZG)\|_F^2 \\ \text{s.t. } & G^T 1_k = 1_m \text{ and } s_{i+1} - s_i \in [1, n_{max}] \end{aligned} \tag{3.5}$$

where $G \in \{0, 1\}^{k \times m}$ is a class indicator matrix and $s \in \Re^{m+1}$ is a vector that contains the start and the end frame of each segment. $dist_\psi^2(Y_i, z_c)$ is the squared distance between the segment Y_i and the centre c and it is expanded as follows

$$dist_\psi^2(Y_i, z_c) = \tau_{ii} - \frac{2}{m_c} \sum_{j=1}^{m} g_{cj} \tau_{ij} + \frac{1}{m_c^2} \sum_{j_1, j_2 = 1}^{m} g_{cj_1} g_{cj_2} \tau_{j_1 j_2} \tag{3.6}$$

where $m_c = \sum_{j=1}^{m} g_{cj}$ is the number of data points (in our case segments) that belong to centre c.

It is worth mentioning that in the special case where each segment is a frame ACA is equivalent to kernel k-means.. ACA seems very similar to kernel k-means but it has four significant differences:

1. ACA clusters can have a different number of features.
2. Constraint for the segment is introduced.
3. ACA uses DTAK kernel.
4. ACA uses Dynamic Programming to solve the minimization problem.

As in DTAK, ACA can be formulated using matrix notation. Firstly, the matrix T can be expressed as follows:

$$T = [tr(K_{ij}^T W_{ij}]_{m\times m} = H(K \circ W)H^T$$

where W is the global correspondence matrix, K is the global kernel frame matrix and H is the segment indicator matrix where $h_{ij} = 1$ if the jth sample belongs to the ith segment.

As mentioned earlier DTAK is not always strictly positive definite kernel. In order to overcome this problem we add a scaled identity matrix to K so that $K \leftarrow K + \sigma I_n$ where σ is absolute value of the smallest eigenvalue of T if it has negative eigenvalues.

So after substituting the optimal value of $Z = [\psi(Y_1), \dots, \psi(Y_m)]G^T(GG^T)^-1$ in (3.5) we have the following form

$$\begin{aligned} J_{aca}(G, H) &= tr((I_m - G^T(GG^T)^{-1}G)T) \\ &= tr((I_m - G^T(GG^T)^{-1}G)H(K \circ W)H^T) \end{aligned} \tag{3.7}$$

3.2.3 Dynamic Programming

ACA can be efficiently solved with the use of coordinate descent optimization where s is computed with the use of dynamic programming and G by using winner-take-all strategy [33].

In every iteration the following subproblem is being solved:

$$G, s = \text{arg min } J_{aca}(G, s) = \text{arg min} \sum_{c=1}^{k} \sum_{i=1}^{m} g_{ci} dist_{\psi}^2(Y_i, z_c) \tag{3.8}$$

To solve this problem dynamic programming is used. With the use of Bellman equation we can write the energy function as follows

$$J(u) = \min_{u-n_{max}<i\leq u}(J(i-1) + \min_{g}\sum_{c=1}^{k} g_c dist^2_{\psi}(X_{[i,u]}, Y_j)) \tag{3.9}$$

where

$$\begin{aligned} dist^2_{\psi}(X_{[i,u]}, z_c) = \tau(X_{[i,u]}, X_{[i,u]}) - \frac{2}{m_c}\sum_{j} = 1^m g_{cj}\tau(X_{[i,u]}, Y_j) \\ + \frac{1}{m_c}\sum_{j_1,j_2=1}^{m} g_{cj_1}g_{cj_2}\tau(Y_{j_1}, Y_{j_2}) \end{aligned} \tag{3.10}$$

In order to solve the optimization problem ACA performs performs the following steps:

1. **Initialization** The first step is the initialization step which can be performed either by using a random initialization, which randomly splits the sequence into segments, or by using propagative segmentation, which initializes the segmentation based on non-tempo-warping clustering.

2. **Iterate** After the initialization step an iterative procedure starts which alternates between a forward and backword step. These steps are described below.

    - **Fordward step** In this step the algorithm starts from the beginning of the sequence until the end and for every position $i-n_{max} < i < u-1$ in the sequence it computes the DTAK between the new segment $X_{[i,u]}$ and all the segments in the previous iteration. In the end, the head position of the segment and the label that have the lowest error $J(u)$ are saved. The algorithm of this step is shown below

```
for $u := 1$ to $n$ do
    $J(u) \leftarrow \infty$
    for $n_u = 1$ to $n_{max}$ do
        Current segment $X_{[i,u]}$ headed by
        $i = u - n_u + 1$;
        for $j = 1$ to $m$do do
            for $s = s_j$ to $s_{j+1} - 1$ do
                Compute $\tau(X_{[i,u]}, Y_j)$ by updating $U(n_u, u, s)$
            end
            $c^* \leftarrow$ arg $\min_c dis_\psi(X_{[i,u]}, z_c)$;
            $J \leftarrow J(i-1) dis_\psi(X_{[i,u]}, z_{c^*})$;
            if $J < J(u)$ then
                $J(u) \leftarrow J, g_u^* \leftarrow e_{u^*}, i_u^* \leftarrow i$;
            end
        end
    end
end
```

Algorithm 1: Forward Step

- **Backword step** In this step the algorithm starts from the end of the sequence and creates the new segments as found in the forward step by changing the head and the label of the segment. This procedure is repeated in backwards. The algorithm for this step is shown below.

```
while $u > 0$ do
    Create a segment $Y = X_{[i_u^*,u]}$ with labels $g_u^*$
    $u \leftarrow i_u^* - 1$;
end
```

Algorithm 2: Backword Step

The time cost for these steps is shown [26] to be $O(n^2 n_{max} t)$, where n is the number of frames, n_{max} is the maximum number of frames that a segment consist of and t is the number of iterations.

3.3 Hierarchical Aligned Cluster Analysis (HACA)

Hierarchical Aligned Cluster Analysis (HACA) [27] is an extension of Aligned Cluster Analysis (ACA) that was described in the previous section. This algorithm belongs to the agglomerative methods of hierarchical clustering, where in these approaches, as described in chapter 1, starts from small clusters and then these are merged together to form larger clusters.

In the first level, HACA first executes ACA with small $n_{max}^{(1)}$ and the DTAK kernel. In later levels another kernel than DTAK is used. This kernel is called Generalized Dynamic Time Alignment Kernel (GDTAK). Given two sequence $X \in \Re^{d \times n_x}$ and $Y \in \Re^{d \times n_y}$ the GDTAK first computes the cumulative kernel matrix as follows

$$u_{ij} = max \begin{cases} u_{i-1,j} + n_i^x \tau_{ij} \\ u_{i-1,j-1} + (n_i^x + n_j^y)\tau_{ij} \\ u_{i,j-1} + n_j^y \tau_{ij} \end{cases} \tag{3.11}$$

where n_i^x is the length of segment $Y_i = X_{[s_i^x, s_{i+1}^x]}$ and τ_{ij} is the DTAK from the previous level. As in DTAK the cumulative kernel matrix is initialized as $u_{11} = (n_1^x + n_1^y)\tau_{11}$. After finding the cumulative kernel matrix the GDTAK is computed as follows

$$\text{v}(X, Y) = \frac{u_{m_x, m_y}}{n_x + n_y} \tag{3.12}$$

To sum up, there are two differences between ACA and HACA:

1. The first difference is that ACA uses DTAK kernel while HACA uses also GDTAK kernel.

2. The second difference is that HACA works in levels while ACA works only in one level. This has an impact in the variable n_{max} where if for example HACA has two levels then it will have $n_{max}^{(1)}$ in the first level and $n_{max}^{(2)}$ for the second level. The equivalent in ACA will be $n_{max} = n_{max}^{(1)} * n_{max}^{(2)}$. So HACA in the first level starts with smaller temporal scales than ACA and this makes it more efficient than ACA. The computational complexity of HACA is $O(n^2 n_{max})$

3.4 ACA - (Graph) Cluster-NMF

In Aligned Cluster Analysis we can use the (Graph) Cluster-NMF with kernel described in previous sections. The kernel that can be used is the DTAK kernel that ACA uses, and the cluster indicator matrix G is ACA's with the addition of a small constant number (preferably 0.2). So the constraint of ACA $G^T 1_k = 1_m$ no longer holds. Thus, we can run the Cluster-NMF and the Graph Cluster-NMF and obtain the cluster indicator matrix G. Then, this matrix can be used to in ACA's objective function. So the minimization problem will have the following form:

$$\begin{aligned} J_{aca} &= \|[\psi(Y_1), \ldots, \psi(Y_m)] - ZG\|_F^2 \\ \text{s.t. } & s_{i+1} - s_i \in [1, n_{max}] \end{aligned} \tag{3.13}$$

In order to make this change possible the dynamic programming search algorithm has to change. More particularly, in every iteration when running the forward step the Cluster-NMF algorithm has to run in order to obtain the cluster indicator matrix G.

Chapter 4

Experiments

In order to test our algorithms thoroughly we run them in a variety of datasets. The Graph Semi- and Convex-NMF algorithms were tested in face and text datasets. The ACA/HACA algorithms were tested in time series datasets. First, we show the results of the Graph NMF algorithms in text and face datasets. Later, we show the results of ACA/HACA algorithms in the time series datasets taken from Imperial College database.

The Graph regularized algorithms can take two parameters: the regularization parameter λ and the number of nearest neighbour p as described in chapter 2. For our experiments we set the number of nearest neighbours to be 5 as in [24] and we experimented with the regularization parameter λ.

4.1 Datasets

The datasets that we run the Graph Semi- and Convex-NMF algorithms are famous text datasets and face datasets.The text datasets are the TDT2, Reuters21578 and RCV1. The face datasets consist of the Yale, ORL and PIE. Finally, the time series datasets are from the Laughter database from the Imperial College London.

The TDT2 dataset consists of 11,201 documents which were collected in 1998 from six different sources. The sub-dataset that we run our algorithms consist of 9,394 and 30 clusters. The Reuters21578 dataset contains 21,578 documents and 135 clusters. This dataset is very popular because it is very difficult for clustering. The subset that we run our algorithms consists of 8,067 documents in total and 65 classes. Finally, the RCV1 dataset consists of 9,625 and 4 clusters. Table 4.1.1 summaries the text dataset.

As for the face datasets, first we run our algorithms in the Yale database which consists of 165 images of size 32×32 and the number of cluster is

	TDT2	Reuters21578	RCV1
no. Documents	9,394	8,067	9,625
no. clusters	30	65	4

Table 4.1.1: Information on the Text Datasets

15. The ORL dataset contains 10 different images which are grouped into 40 clusters. The PIE dataset includes 41,368 images of size 32×32 and the number of clusters are 68. Table 4.1.2 summaries the face datasets.

	Yale	ORL	PIE
no. Images	165	400	41,368
no. clusters	15	40	68

Table 4.1.2: Information on the Face Datasets

ACA/HACA algorithms were run in time series datasets. These datasets comes from the Laughter database of the Imperial College London. We used 8 datasets to run the algorithms from 7 different subjects. These datasets contains two classes: laugh or no laugh.

4.2 Evaluating clustering algorithms

In order to evaluate our clustering algorithms we used one metric. This was the accuracy. This metric is defined as follows:

Given a dataset $D \in \Re^{n \times d}$ and data point x_i in the dataset the accuracy is defined as follows:

$$Accuracy = \frac{\sum_{i=1}^{n} \delta(map(l_i), g_i)}{n}$$

where n is the total number of data points in the dataset, l_i is the label of our cluster algorithms, g_i is the ground truth labels and $map(l_i)$ is the best map between the l_i and the g_i. Tha δ function is defined as follows:

$$\delta(x, y) = \begin{cases} 1 & \text{if x = y} \\ 0 & \text{otherwise} \end{cases}$$

4.3 Comparisons

Before illustrating our results, we should mention the comparisons that we make between the Graph regularized algorithms. The first comparison is between the GNMF and the Graph Semi- and Convex- NMF algorithm, that is with the regularization parameter. The second comparison is between the Semi-NMF and the Graph Semi-NMF algorithms as well as between the Convex-NMF and the Graph Convex-NMF algorithms. The Semi- and Convex-NMF algorithms is basically when the regularization parameter is set to zero.

As for the dynamic data, the comparison of the algorithms are between the ACA/HACA algorithms and the variations with the Cluster-NMF and the Graph Cluster-NMF algorithms.

4.4 Results

In this section we are going to show the results of the algorithms in different datasets. In the first part, we show the results of the Graph regularized algorithms in numerous dataset. In the second part, we run the ACA/HACA algorithms with the Laughter data and we present the results.

4.4.1 Static Data

The demonstration of the results is with the help of tables and figures, and the highest value is highlighted. For the regularization parameters of the Graph NMF algorithms we used the values 0, 5 ,10 ,50 ,100 and 130 in most cases. There was one case where we used more values because the results were not clear. The nearest neighbour parameter p was set to 5. Because the results of the algorithms are stochastic, we run the algorithms ten times for each value of λ and we took the mean accuracy.

The first dataset we run the algorithm was the TDT2. Table 4.4.1 and Figure 4.4.1 show the results.

From the results we can conclude that the Graph Semi-NMF algorithm performs better results than the other two algorithms (Graph NMF and Graph Convex-NMF) with its highest value to be 83.33%. Moreover, the Graph Convex-NMF very poor results with its highest value to be 45% when the regularization parameter λ is 10. For the Graph-NMF algorithm the highest value is when λ is 100, while for the Graph Semi-NMF when λ is 10. Furthermore, when $\lambda = 0$, that is when the algorithms NMF, Semi-NMF and Convex-NMF run, each algorithm performs its worst. With the

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	45,83	47.08	39.77
10	78,12	**83.33**	45.81
50	**81,15**	80.12	**45.89**
100	79,27	80.17	43.60
130	79,89	79.99	43.72

Table 4.4.1: Accuracy(%) results on TDT2 Text Dataset

regularization parameter not set to zero, all the algorithms perform better with the Graph-NMF and Graph Semi-NMF to increase significally with 35.32 and 36.25, respectively.

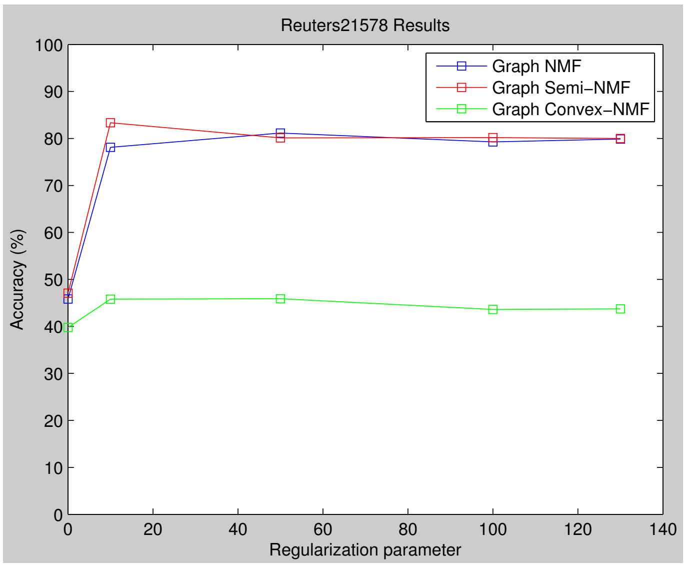

Figure 4.4.1: TDT2 Results

The second text dataset we run the algorithms was the Reuters21578. Table 4.4.2 and Figure 4.4.2 show the results. From the results we obtained we can see that Graph NMF and Graph Semi-NMF perform better than NMF and Semi-NMF with approximately 10% and 13%, respectively. Moreover, the Graph Convex-NMF is performing a little better than the Convex-NMF with 1.09%. Furthermore, the Graph Semi-NMF algorithm performs better than the Graph NMF and the Convex-NMF algorithm.

The last text dataset we run was the RCV1. Table 4.4.3 and Figure 4.4.3 show the results. From the results we can deduce that the regularization

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	17.27	17.92	13.37
10	25.82	30.51	11.73
50	**27.99**	**30.85**	14.05
100	26.55	29.67	**14.46**
130	26.79	30.86	11.97

Table 4.4.2: Accuracy(%) results on Reuters21578 Text Dataset

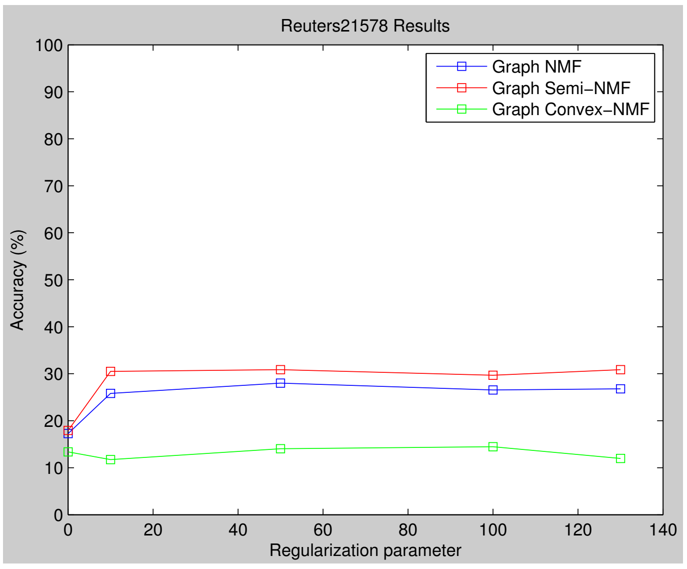

Figure 4.4.2: Reuters21578 Results

parameter does not improve the Graph NMF and Graph Semi NMF algorithms, rather the accuracy has a high drop. We can see that the accuracy drops by approximately 30% for both algorithms. So the NMF, Semi-NMF algorithms (when $\lambda = 0$) have better results. On the other hand, for the Graph Convex NMF algorithm the accuracy seems to remain unchanged. It increases about 0.83%, when $\lambda = 50$, and it is the highest from all the algorithms. So the Graph Convex-NMF is performs better results than the Convex-NMF algorithm (when $\lambda = 0$).

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	**63.19**	**59.22**	70.06
10	32.67	32.93	68.58
50	31.95	31.72	70.89
100	31.67	32.77	69.53
130	31.35	33.02	**71.16**

Table 4.4.3: Accuracy(%) results on RCV1 Text Dataset

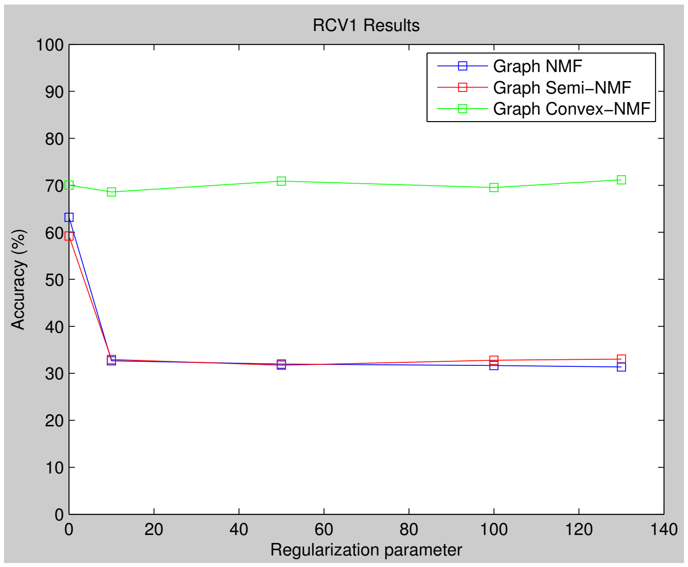

Figure 4.4.3: RCV1 Results

RCV1 was the last text dataset that we run our algorithms. Now we are going to present the results of our algorithms for the face datasets. The first dataset we experiment with was the Yale dataset from the Yale database. Table 4.4.4 and Figure 4.4.4 show the results.

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	**40.97**	**37.64**	25.46
10	26.55	34.73	32.00
50	27.27	33.64	**34.42**
100	27.63	32.73	34.06
130	28.06	32.30	33.88

Table 4.4.4: Accuracy(%) results on Yale Face Dataset

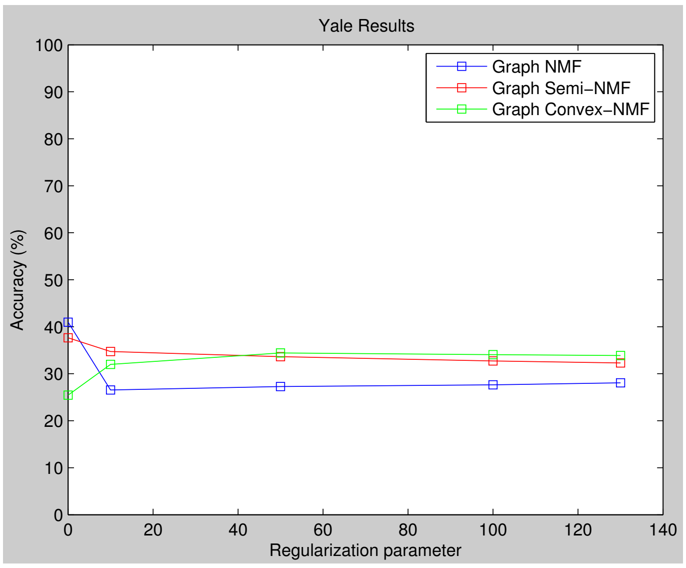

Figure 4.4.4: Yale Results

From the results we can conclude that NMF performs better results than the Graph NMF, with approximately 15% better accuracy. The Semi-NMF algorithm also performs better results than the Graph Semi-NMF algorithm, with approximately 3% better accuracy. On the other haand, the Graph Convex-NMF performs better results than the Convex-NMF, with approximately 9% better accuracy. Moreover, the Graph Semi-NMF and the Graph Convex-NMF perform better results than the Graph NMF, with the Graph Convex-NMF to have the best results. All in all, for this dataset the NMF algorithm performs the best.

The second face dataset, we run our algorithm was the ORL dataset. For this dataset we could not conclude which algorithm performs best for the values of the regularization parameter that we run our algorithm. That is why we run our algorithms for more values of regularization parameter. Table 4.4.5 show the results.

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	48.93	35.70	15.88
10	48.35	**56.45**	18.45
50	**50.10**	53.83	38.80
100	47.90	50.20	47.43
130	48.08	53.43	50.18
150	48.85	52.90	50.95
200	48.70	51.58	52.80
250	49.60	52.20	54.08
300	48.28	52.23	56.25
350	48.18	52.03	55.93
400	49.78	53.18	**58.10**
450	49.35	51.28	57.50

Table 4.4.5: Accuracy(%) results on ORL Face Dataset

From Table 4.4.5 we can conclude that the Graph NMF algorithm performs better than the NMF algorithm with approximately 2%. Moreover, the Graph Semi NMF algorithm performs a lot better than the Semi NMF algorithm with approximately 21%. The Graph Convex NMF algorithm also overwhelms the Convex-NMF algorithm with approximately 42%. Overall, the Graph Convex-NMF performs the best among the algorithms. It gives better results than the Graph Semi-NMF by $1,65\%$ and 8% better than Graph NMF. The best value for the regularization parameter for the Graph NMF was 50, for the Graph Semi-NMF was 10 and for the Graph Covnex-

NMF was 400. We can see that the Graph Convex-NMF needed a lot bigger value of the regularization parameter than the other algorithms.

Figure 4.4.5 also shows the results, where we can see that the Graph NMF is more or less keeping the same accuracy around 50%, the Graph Semi-NMF is increasing its accuracy and then its being unchanged for accuracy a little above 55%. Finally, the Graph Convex-NMF seems to exponentially increase its accuracy until is is stable around 55% .

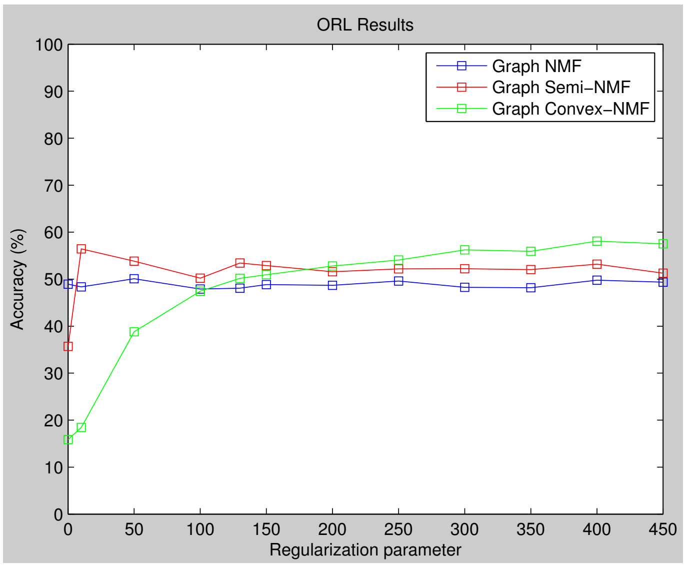

Figure 4.4.5: ORL Results

The last face dataset we run the algorithms was the PIE dataset. Table 4.4.6 show the results. For this dataset the Convex-NMF algorithm run only once for each regularization parameter because of the large computational time it takes.

λ	Graph NMF	Graph Semi-NMF	Graph Convex-NMF
0	12.54	11.75	3.94
10	17.74	18.31	**3.96**
50	18.06	**18.48**	3.83
100	**18.18**	18.45	3.92
130	17.62	18.09	3.82

Table 4.4.6: Accuracy(%) results on PIE Face Dataset

From Table 4.4.6 we can deduce that the Graph NMF and Graph Semi-NMF algorithm improves the NMF and the Semi-NMF algorithms, respectively, by approximately 6%. The Graph Convex-NMF seems to have more or less the same poor results as the Convex-NMF. Moreover, the Graph Semi-NMF has slightly better results than the Graph NMF algorithm with 0.3% better results. It is worth mentioning that among all the regularization parameters the Graph Semi-NMF algorithm has better results than the Graph NMF, while the Semi-NMF algorithm has worse results than the NMF algorithm. The results of the table are also shown in Figure 4.4.6, as a curve.

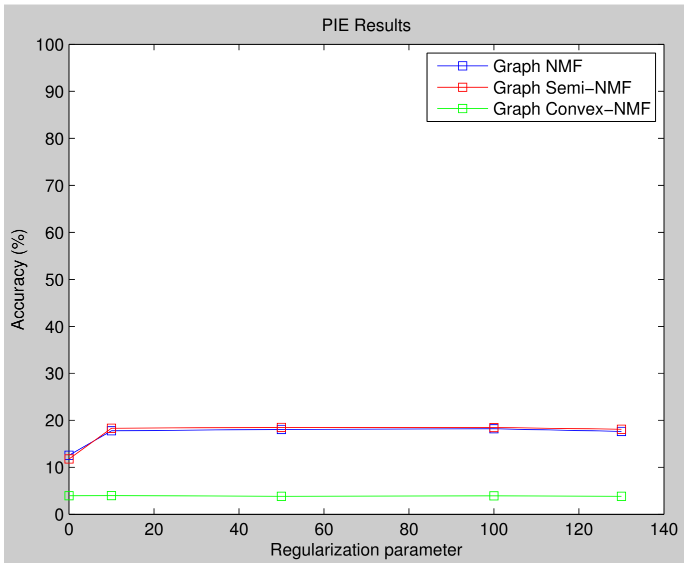

Figure 4.4.6: PIE Results

In order to make more clear about the performance of the algorithms, we will compare the Semi-NMF with the best performance of Graph Semi-NMF and the Convex-NMF and Graph Convex-NMF algorithm.

Figure 4.4.7 show the comparison of Semi-NMF with Graph Semi-NMF along of the datasets. From the figure we can see that Graph Semi-NMF has better results in 4 out of 6 datasets than Semi-NMF. For the TDT2, ORL and PIE datasets the accuracy is more than 10%. For the RCV1 and Yale dataset the Semi-NMF algorithm performs better than the Graph Semi-NMF algorithm with the RCV1 dataset to have a lot higher accuracy.

Figure 4.4.8 shows the comparison of Convex-NMF with Graph Convex-NMF along of the datasets. From the results we can conclude that Graph Convex-NMF performs better than the Convex-NMF algorithm in all the datasets. The datasets where it did a lot better were the TDT2, Yale and the ORL with the difference to range from 6.12% to 42.22%.

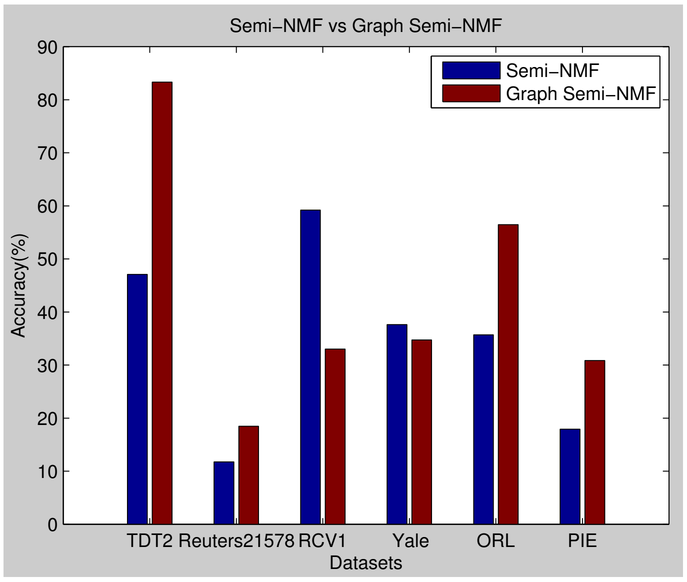

Figure 4.4.7: Semi-NMF vs Graph Semi-NMF

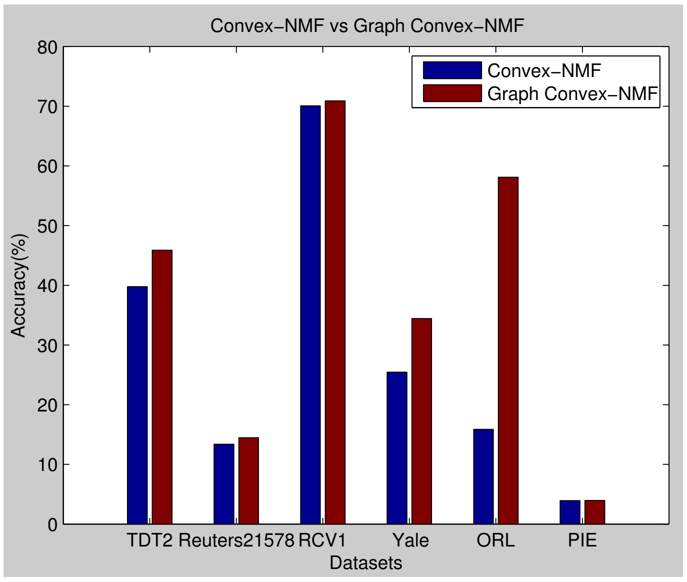

Figure 4.4.8: Convex-NMF vs Graph Convex-NMF

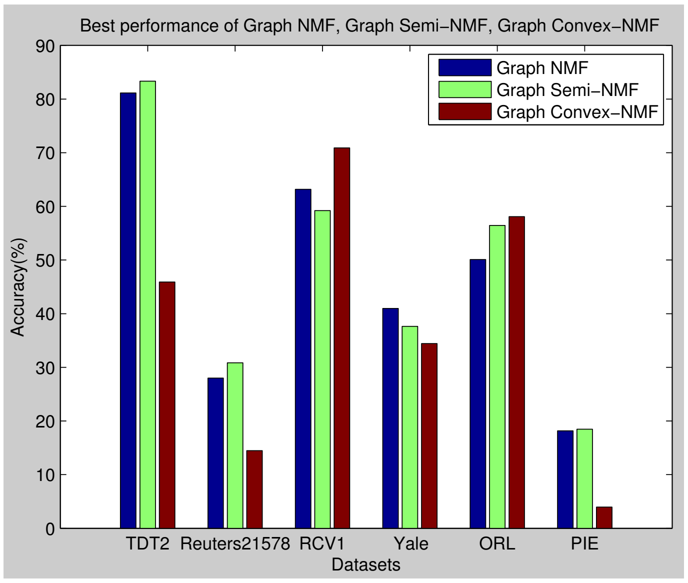

Figure 4.4.9: Comparison of all algorithms

In the last Figure we present the best performance of each algorithm in the six datasets we run (Figure 4.4.9). We can see that only in the Yale dataset the Graph NMF algorithm performs better than the Graph Semi- and Graph Covex-NMF algorithms. Moreover, the Graph Semi-NMF algorithm performs better in 3 out of 6 datasets. Finally, the Graph Convex-NMF performs better in 2 out of 6 datasets.

4.4.2 Dynamic Data

Now we are going to present the ACA/HACA results on the Laughter data of Imperial College database. We show how the original ACA/HACA algorithms perform and how they perform using the Cluster-NMF and the Graph Cluster-NMF algorithms.

Table 4.4.7 and Figure 4.4.10 show the results in the different dataset. In the Figure the ACAC means the ACA algorithm with the Cluster-NMF and the ACAGC means the ACA algorithm with the Graph Cluster-NMF.

From the results we can see that in two datasets ACA performs better than the two variations. More particularly, the ACA with Cluster-NMF algorithm performs better or equal in 6 out of 8 datasets than the ACA algorithm. Moreover, the ACA with the Graph Cluster-NMF algorithm performs better or equal in also 6 out of 8 datasets than the ACA algorithm.

Dataset	ACA	ACA - CNMF	ACA - GCNMF
S001-003	**73**	67	66
S002-001	52	**79**	78
S002-008	91	**93**	91
S003-005	52	**90**	65
S005-005	56	**93**	92
S006-004	68	68	68
S008-001	90	90	90
S011-001	**56**	52	53

Table 4.4.7: Accuracy(%) results for ACA algorithm

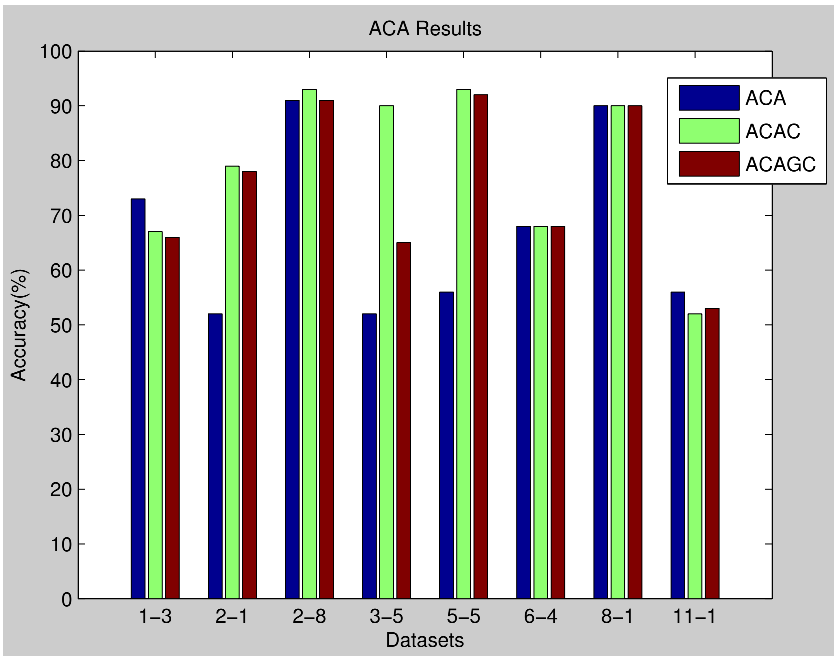

Figure 4.4.10: ACA Results

Now we present the HACA results in Table 4.4.8 and Figure 4.4.11. From the results, we can conclude that the Cluster-NMF with the HACA algorithm performs better or equal in 5 out of 8 datasets than the plain HACA algorithm. Moreover, the Graph Cluster-NMF with HACA algorithm performs better or equal in 7 out of 8 datasets than the HACA algorithm.

Dataset	HACA	HACA - CNMF	HACA - GCNMF
S001-003	**73**	67	66
S002-001	52	77	**78**
S002-008	**91**	89	**91**
S003-005	51	**88**	59
S005-005	91	**92**	**92**
S006-004	68	89	68
S008-001	**90**	89	**90**
S011-001	66	**90**	89

Table 4.4.8: Accuracy(%) results for HACA algorithm

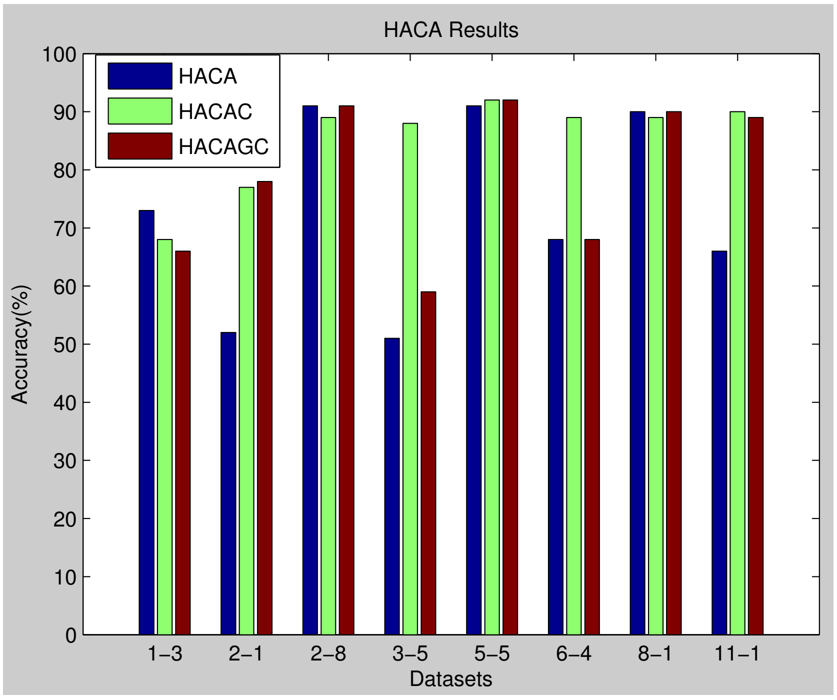

Figure 4.4.11: HACA Results

Chapter 5

Conclusions and Future work

In this chapter, we present our conclusions for the algorithms we implemented. Moreover, we discuss how further investigation on the evaluation and the performance of the algorithms can be done as a future work.

5.1 Conclusions

This project presented two new methods for clustering that extended the Semi- and Convex-NMF. In particular, we propose the Graph Semi- and Graph Convex-NMF. These methods are a further improvement for the Semi- and Convex-NMF, in that they take into account the intrinsic geometry of the data by calculating the Laplacian matrix.

As shown in the previous chapter, these methods perform, in most datasets, better than the Semi- and Convex-NMF techniques. More particularly, the Graph Semi-NMF performs better in 4 out of 6 dataset and the Graph Convex-NMF performs better in all the dataset than the Convex-NMF. In addition, our algorithms perform better than the GNMF algorithm in 5 out of 6 datasets.

From the results we can see that the restriction of NMF, for the nonnegativity of the matrix centroid F, limits its performance in most dataset. That is why Graph Semi-NMF algorithm, which does not have this constraint, performs better. Moreover, Graph Convex-NMF assumptions (that the centroid matrix is a convex combination of the data) seem to hold in RCV1 and ORL datasets. That is why the results are better than the other two algorithms. However, this is not the case for the other four datasets.

The next tests we tried, we combined the Cluster- and Graph Cluster-NMF with the ACA/HACA algorithms and run them in time series datasets. The variations of ACA performed better in most datasets than ACA itself.

Same results were concluded with HACA algorithm where the variations of HACA performed better in most datasets. In addition, we can conclude that the Cluster-NMF and Graph Cluster-NMF improved ACA more than the HACA algorithm.

From these results, we can conclude that softening the assignment of the clustering in ACA/HACA algorithms the performance is improved. First, ACA/HACA algorithms used the K-means algorithm for the initialization of the cluster indicator matrix G. Now the initialization is achieved by running the Cluster-NMF and Graph Cluster-NMF algorithm. The improvement in the performance of the algorithms was expected as the Semi-NMF and Convex-NMF techniques tend to perform better than K-means algorithm.

5.2 Future Work

Further research can be made in our work. More particularly, Graph NMF algorithms have two kinds of parameters. The first one is the number of nearest neighbours that will be included to create the weight matrix and the second is the regularization parameter. In this project the nearest neighbour parameter was fixed and the regularization parameter was set to different values in order to see which value performs the best. A more appropriate technique to investigate these two parameters and see which two perform best is to use hill climbing technique. Using this technique the two parameters can be investigated fast and find the optimum numbers. The disadvantage of this technique is that this optimum may not be the global optimum rather a local optimum. Another technique that can be used is the cross validation for model selection. In this technique one parameter at a time can be used and evaluate while keeping the other fixed. This technique is very slow.

Evaluation of the algorithms can be made using cross validation. It would be interesting to do a 10-fold cross validation to the algorithms and record their performance. Not only that, after the cross validation a hypothesis testing can be used to compare the algorithms. One such test can be the paired t-Test. To run this test the folds in each algorithm run should be the same.

Besides the evaluation of the algorithms, a further investigation can be made in the initialization of the algorithms. For this thesis the initialization was random. Other initializations that can be used are a K-means or with agglomerative algorithm. It would be interested to see how the initialization affects the performance of the algorithms.

Chapter 6

User Guide

In this chapter, we describe how our code can be used. First of all, we should mention that the ACA/HACA and Graph NMF code is not implemented by us. ACA/HACA code was provided by F. De la Torres web page [34] and the Graph NMF code was provided by D. Cai web page [35]. The Semi-NMF, Convex-NMF, Cluster-NMF and Kernel-NMF algorithms were implemented by us. All the folders, contain a Readme.txt file where short instructions are given on how to run the algorithms.

6.1 ACA/HACA Code

ACA/HACA code is provided in the "aca" folder. The code that is provided contains the ACA/HACA algorithm along with the GMM algorithm. In the folder of the code there are the following folders:

- data: This folder contains some data that were provided with the code.
- lib: This folder contains the library of the functions that are used.
- oldTracker: Contains the data from the Laughter database that we run our algorithm
- src: Contains the source files of the algorithms

Besides these folders there are some files that are can run the algorithms. Before running the algorithm the make.m and addPath.m files should be run.The main file we used is the demoMocap, which was modified by us in order to be able to run our data files. In that file you can change the files that are to be run.

Another useful file is the acaFordSlow.m which we also modified in order to be able to run the Cluster-NMF and the Graph Cluster-NMF algorithms. This file is located in ./aca/src/seg/aca folder. There you can comment the code we inserted if you want to run just the ACA/HACA algorithms. If you want to run ACA/HACA with the Cluster-NMF algorithm then the code should be uncomment and the option.alpha variable should be set to zero. If you want to run the Graph Cluster-NMF algorithm then you should initialize the regularization parameter alpha not to be zero. We run the Graph Cluster-NMF with alpha equal to ten.

6.2 Graph NMF Code

In this folder we provide the face and text datasets that we run our algorithms in the "Datasets" folder. The main file to run is the example.m. In this file the desirable options to run the algorithm can be set. The most important attributes of the options struct variable that can be set are the following:

- alg: Specifies the algorithm to run. Choices: "nmf","seminmf","convexnmf" and "clusternmf"
- WeightMode: Specifies how to construct the weight matrix. Choices: "Binary","HeatKernel" and "Cosine"
- error: The error between two successive iterations.
- maxIter: The maximum number of iterations the algorithm can run.
- minIter: The minimum number of iterations the algorithm can run.
- nRepeat: The number of repeats of the algorithm. The value must be between the value of maxIter and minIter.
- meanFitRatio: The parameter specifies the step of the algorithm.
- alpha: The regularization parameter.

We also modified the GNMF_Multi.m file where we included the Graph Semi-NMF, the Graph Convex-NMF and the Graph Cluster-NMF.

6.3 NMF Code

The files in the NMF folder were implemented by us. Important files in the folder are the following:

- semiNMF: Contains the Semi-NMF algorithm.
- convexNMF: Contains the Convex-NMF algorithm.
- GeneralFile: This file is a general file where the algorithms can be run.

In the GeneralFile.m file the options parameter can take the following values:

- random: Takes two values "true" and "false". If the value is set to "true" then the initialization of the cluster indicator matrix G is random, otherwise the initialization is done with K-Means.
- maxIterations: The maximum number of the iterations the algorithm will run.
- error: The value $\|X - FG^T\|^2$
- betweenErros: The difference between the errors of two succesive iterations.

Bibliography

[1] J. Han, M. Kamber *"Data Mining: Concepts and Techniques",* Morgan Kaufmann, San Francisco, pp. 346389, 2001.

[2] J. MacQueen *"Some methods for classification and analysis of multivariate observations",* Proc. Fifth Berkeley Symp. on Math. Statist. and Prob., Vol. 1, 1967

[3] L. Kaufman, P.J. Rousseeuw *"Finding Groups in Data: An Introduction to Cluster Analysis",* Wiley, New York, 1990.

[4] J.C. Bezdek *"Pattern Recognition with Fuzzy Objective Function Algorithms",* Plenum Press, New York and London,1987.

[5] R. Krishnapuram, A. Joshi, O. Nasraoui, L. Yi *"Low complexity fuzzy relational clustering algorithms for web mining",* IEEE Trans. Fuzzy Systems 9 (4), pp. 595607, 2001.

[6] M.Ester, H.-P. Kriegel, J. Sander, X. Xu *"A densitybased algorithm for discovering clusters in large spatial databases",* Proceedings of the 1996 International Conference on Knowledge Discovery and Data Mining (KDD96), Portland, OR, pp. 226231, 1996.

[7] M. Ankerst, M. Breunig, H.-P. Kriegel, J. Sander *" OPTICS: ordering points to identify the clustering structure",* In Proc. of the 1999 ACM-SIGMOD Int. Conf. on Management of Data, Philadelphia, PA, pp. 4960,1999.

[8] W. Wang, J. Yang, and R. Muntz *" STING : A Statistical Grid Appraoch to Spatial Data Mining :",* In Proc. 1997 Int. Conf. Very Large Data Bases (VLDB97), pages 186-195, Athens, Greece, Aug. 1997.

[9] R. Agrawal,J. Gehrke, D. Gunopoulos, and P. Raghavan *"Automatic subspace clustering of high dimensional for data mining applications",* In Proc. of the 1998 ACM SIGMOM Int. Conf. on Management of data, pp.94 - 105, ACM Press, 1998

[10] T.Warren Liao, *"Clustering of time series data—a survey",* Pattern Recognition, 2005

[11] C.S. Moller-Levet, F. Klawonn, K.-H. Cho, O. Wolkenhauer, *"Fuzzy clustering of short time series and unevenly distributed sampling points",* In Proc. of the 5th Int. Symposium on Intelligent Data Analysis, Berlin, Germany, August 2830, 2003.

[12] I.S. Dhillon, Y. Guan, B. Kulis, *"Kernel k-means, Spectral Clustering and Normalized Cuts",* KDD04, August 22-25, 2004

[13] T. Kohonen *"The self organizing maps",* Proc. IEEE, vol. 78,issue 9, pp.14641480, 1990.

[14] P. Paatero and U. Tapper, *"Positive matrix factorization: a non-negative factor model with optimal utilization of error estimates of data values.",* Environmetrics, 5(1):111126, 1994

[15] W. Xu, X. Liu, and Y. Gong *"Document clustering based on non-negative matrix factorization",* in Proc. ACM Conf. Research development in IR(SIRGIR), pp. 267 - 273 2003.

[16] V. P. Pauca, F. Shahnaz, M.W. Berry, R.J. Plemmons *"Text mining using Non-Negative Matrix Factorizations",* in Proc. of 4th SIAM International Conference on Data Mining, 2004.

[17] H. Kim, H.Park *"Sparse non-negative matrix factorization via alternating non-negativity-constrained least squares for microarray data analysis",* Bioinformatics, vol.23, no12, pp. 1495 - 1502, 2007.

[18] D. Greene, G. Cagney, N. Krogan, and P. Cunningham *"Ensemble non-negative matrix factorization methods for clustering protein-protein interactions",* Bioinformatics, vol.24, no15, pp. 1722 - 1728, 2008.

[19] P. Smaragdis and J. C. Brown. *"Non-negative matrix factorization for polyphonic music transcription",* In IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, pages 177-180, 2003.

[20] F. Sha and L. K. Saul. *" Real-time pitch determination of one or more voices by non-negative matrix factorization",* In Proc. NIPS2004, pages 1233-1240. MIT Press, 2005.

[21] D.D. Lee, H.S. Seung *"Algorithms for Non-negative Matrix Factorization",* in Advances in Neural Information Processing Systems 13. Cambridge, MA: MIT Press, 2001.

[22] D.D. Lee, H.S. Seung *"Learning the parts of objects by non-negative matrix factorization",* Nature, vol. 401, pp.788 - 791, 1999.

[23] Chris H. Q. Ding, Tao Li, Michael I. Jordan *"Convex and Semi-Nonnegative Matrix Factorizations",* IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, pp. 44-55, 2010.

[24] D. Cai, X. He, J.Han *"Graph Regularized Non-negative Matrix Factorization for Data Representation",* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 33, No. 8, pp. 1548-1560, 2011.

[25] M. Belkin and P.Niyogi *"Laplacian eigenmaps and spectral techniques for embedding and clustering",* In Advances in Neural Information Processing Systems 14,pp. 585 - 591, MIT Press, Cambridge, MA, 2001

[26] F. Zhou, F. De la Torre, and J.K. Hodgins, *"Aligned Cluster Analysis for Temporal Segmentation of Human Motion",*. Proc. Eighth IEEE Int'l Conf. Automatic Face & Gesture Recognition, 2008.

[27] F. Zhou, F. De la Torre, and J.K. Hodgins, *"Hierarchical Aligned Cluster Analysis for Temporal Clustering of Human Motion",*. IEEE Trans. Pattern Anal. Mach. Intell., pp.582-596, 2013.

[28] H.Shimodaira, K.-I. Noma, M. Nakai, and S. Sagayama, *"Dynamic Time-Alignment Kernel in Support Vector Machine",*. Proc. Neural Information Processing Systems, 2001.

[29] L.Rabiner, B. Juang *"Fundamental of Speech Recognition",*. Prentice Hall, 1993

[30] B. Haasdonk, *"Feature Space Interpretation of SVMs with Indefinite Kernels",*. IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 27, no. 4, pp. 482-492, Apr. 2005

[31] M. Cuturi, J.-P. Vert, O. Birkenes, and T. Matsui, *"A kernel for Time Series Based on Global Alignments",*. Proc. IEEE Int'l Conf. Accoustics, Speech, and Signal Processing, 2007.

[32] B. Scholkopf and A.J. Smola, *"Learning with Kernels",*. MIT Press, 2002

[33] S.Roweis and Z. Ghahramani, *"A unifying Review of Linear Gaussian Models",*. Neural Computation, vol. 11, no. 2, pp. 305-345, 1999

[34] http://www.f-zhou.com/tc_down.html

[35] http://www.cad.zju.edu.cn/home/dengcai/Data/GNMF.html

[36] M. Belkin and P. Niyogi *"Laplacian eigenmaps for dimensionality reduction and data representation.",* Neural Computation,15(6),pp. 13731396,2003.

[37] Y. Papadakis, C. Kotropoulos and G. R. Arce *"Non-Negative Multilinear Principal Component Analysis of Auditory Temporal Modulations for Music Genre Classification",* IEEE Trans. on Audio, Speech, and Lang. Proc., vol. 18, no. 3, 2010

[38] Z. Yang and J. Laaksonen, *"Multiplicative updates for non-negative projections",* Neurocomputing, vol. 71, no. 1-3, pp. 363-373,2007

[39] J.Yoo and S. Choi, *"Orthogonal nonnegative matrix factorization: Multiplicative updates on Stiefel manifolds",* In Proc, 9th Int. Conf. Intell. Data Eng. and Autom. Learn. (IDEAL), 2008

Appendix A

Mathematical Preliminaries

In this section we are going to discuss some mathematical preliminaries that are necessary to understand the algorithms that we present in depth.

A.1 Laplacian eigenmaps

Laplacian eigenmaps [36] are used for dimensionality reduction. The definition of the dimensionality reduction problem is the following:

Definition 3. *Given the data points $x_1, x_2, \ldots, x_n \in \Re^l$ we want to find a mapping such that the points $y_1, y_2, \ldots, y_n \in \Re^m$ that belong to a subspace $m << l$, represent the data points best.*

Laplacian eigenmaps take into consideration the intrinsic geometry of the data. That is, if the data points x_i, x_j are "close" to each other in the original space then they are also "close" to the subspace. This is achieved by constructing a graph where the nodes are the data points such that the geometry is taken into account. More particularly, the steps of the algorithm are described below.

1. **Constructing the Graph**

   The first step is to construct a graph for the data points. The data points represent the nodes of the graph and they can be connected to each other with the following two methods.

   Threshold neighbourhood

   For this method, a threshold ϵ is used and a node (data point) x_i is connected to a point x_j if $\|x_i - x_j\|^2 < \epsilon$.

K nearest neighbour

For this method we choose to connect a node x_i with each K nearest neighbours.

2. **Choosing weights for the edges**

The second step is to choose appropriate weights for the edges of the graph. This can be done with the following two methods.

Using Heat kernel

Given that the nodes x_i and x_j are connected the Heat kernel weight is computed as follows

$$W_{ij} = \exp^{-\frac{\|x_i - x_j\|^2}{t}}$$

where t is a parameter specified by the user.

Binary

If x_i is connected to x_j then the weight is set to 1. If two nodes are not connected then the value is set to 0. This weight is similar as taking the heat kernel matrix with $t \to \infty$.

3. **Eigenmaps**

Now that the graph of the data has been created, the last step is to find the appropriate mapping for our data. In order to do that we define

$$L = D - W$$

where W is the weight matrix for the edges as computed in Step 2 and D is a diagonal matrix and it is defined to be $D_{ii} = \sum_j W_{ji}$. L is known as the Laplacian matrix and it is symmetric. Now that we have defined the Laplacian matrix the subspace can be found by solving the following eigenvector problem.

$$Lf = \lambda Df$$

where λ are the eigenvalues and f are the eigenvectors. For this problem the solution for $\lambda = 0$ is not taken into account. Suppose the eigenvector corresponding to this eigenvalue is f_0, then we take the next m eigenvectors $f_1, f_2, \ldots, f_m$ to construct the Euclidean subspace, where the corresponding eigenvalues for these eigenvectors are $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_m$.

A.2 Multiplicative Update Rules

In this section, we describe the multiplicative update rules and how they can derived. Suppose we want to minimize the following optimization problem $V^* = \arg\min_{V \geq 0} f(V)$ using the natural gradient ascent algorithm. Given the function f that we want to minimize the form of the gradient algorithm is the following

$$V^{t+1} = V^t - \eta \nabla_V f$$

To preserve the non-negativity of the function, we use the strategy described in [37],[38] and [39]. In these papers, the natural gradient is decomposed into two non-negative parts. That is,

$$[\nabla_V f]_{i,j} = [\nabla_V f]^+_{i,j} - [\nabla_V f]^-_{i,j}$$

where

$$(\nabla_V f)^+_{i,j} = \begin{cases} (\nabla_V f)_{i,j} & \text{if } (\nabla_V f)_{i,j} > 0 \\ 0 & otherwise \end{cases}$$

$$(\nabla_V f)^-_{i,j} = \begin{cases} -(\nabla_V f)_{i,j} & \text{if } (\nabla_V f)_{i,j} < 0 \\ 0 & otherwise \end{cases}$$

According to [37] the step size η can be data-dependent. That is,

$$\eta = (\frac{V^t}{\nabla_V f^+})_{i,j}$$

Then, the multiplicative rule for the parameter V is the following

$$\begin{aligned} V^{t+1} &= V^t - [\frac{V^t}{\nabla_V f^+}]_{i,j}([\nabla_V f]^+_{i,j} - [\nabla_V f]^-_{i,j}) \\ &= V^t \frac{\nabla_V f^-}{\nabla_V f^+ + \epsilon} \end{aligned}$$

This is the procedure we use when we derive the rules for the NMF algorithms with the non-negativity constraint.