Efficient Kernel-based Clustering

Grigorios Tzortzis

MASTER THESIS

—♦—

Ioannina, September 2008

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF IOANNINA

Αποδοτικές Μέθοδοι Ομαδοποίησης με Χρήση Συναρτήσεων Πυρήνα

Η ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης

του Τμήματος Πληροφορικής Εξεταστική Επιτροπή

από τον

Γρηγόριο Τζώρτζη

ως μέρος των Υποχρεώσεων για τη λήψη του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ

ΣΤΙΣ ΤΕΧΝΟΛΟΓΙΕΣ - ΕΦΑΡΜΟΓΕΣ

Σεπτέμβριος 2008

Acknowledgements

This thesis was elaborated as part of the postgraduate studies program of the Computer Science Department of the University of Ioannina in Greece under the supervision of associate professor Aristidis Likas who I thank for our great cooperation, his useful comments and support.

Part of this work has been supported by Interreg IIIA Greece-Italy 2000-2006 (Interstore, grant I2101005).

Table of Contents

List of Figures

List of Tables

List Of Algorithms

Abstract

Tzortzis, Grigorios, F.
MSc, Computer Science Department, University of Ioannina, Greece. September, 2008.
Thesis Title: Efficient Kernel-based Clustering.
Thesis Supervisor: Aristidis Likas.

This thesis studies the clustering problem which aims to partition a dataset into a finite number of groups (clusters) such that data points in the same cluster are similar to each other and dissimilar to those of other clusters. Clustering has found application in a variety of fields such as machine learning, pattern recognition, image segmentation, spatial database analysis, life and medical sciences, economics and many more.

In this work, we focus on kernel-based clustering methods and in particular on the Kernel k-Means algorithm which is an extension of the standard k-Means clustering algorithm that identifies nonlinearly separable clusters. In order to overcome the cluster initialization problem associated with this method, we propose the Global Kernel k-Means algorithm, a deterministic and incremental approach to kernel-based clustering. Our method adds one cluster at each stage through a global search procedure consisting of several executions of Kernel k-Means from suitable initializations. This algorithm does not depend on cluster initialization, identifies nonlinearly separable clusters and, due to its incremental nature and search procedure, locates near optimal solutions avoiding poor local minima. Furthermore two modifications are proposed to reduce the computational cost that do not significantly affect the solution quality.

Based on the weighted Kernel k-Means algorithm, a modification of the Kernel k-Means algorithm that assigns different weights to each data point, we discuss how to alter the Global Kernel k-Means algorithm and its variants in order to accommodate weights. The use of weights is very important in proving an equivalence between the objective of these methods and many popular graph cut criteria.

We test the proposed methods on artificial data, face images, handwritten digits, graphs and for the first time we employ Kernel k-Means for MRI segmentation along with a novel kernel. The proposed methods compare favorably to Kernel k-Means with random restarts.

Εκτεταμενη Περιληψη στα Ελληνικα

Γρηγόριος Τζώρτζης του Φωτίου και της Αθανασίας.
MSc, Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Σεπτέμβριος, 2008.
Τίτλος Διατριβής: Αποδοτικές Μέθοδοι Ομαδοποίησης με Χρήση Συναρτήσεων Πυρήνα.
Επιβλέπων: Αριστείδης Λύκας.

Στην εργασία αυτή μελετάται το πρόβλημα της ομαδοποίησης (clustering). Στόχος της ομαδοποίησης είναι να διαχωρίσει ένα σύνολο δεδομένων σε ομάδες (clusters) τέτοιες ώστε η ομοιότητα ανάμεσα στα δεδομένα μίας ομάδας να είναι μεγαλύτερη από την ομοιότητα μεταξύ των δεδομένων διαφορετικών ομάδων. Η ομαδοποίηση βρίσκει εφαρμογή σε πολλά πεδία όπως αυτά της αναγνώρισης προτύπων, της μηχανικής μάθησης, της εξόρυξης δεδομένων, της επεξεργασίας εικόνας, της βιολογίας κ.α.

Στα πλαίσια αυτής της διατριβής επικεντρωνόμαστε σε μεθόδους ομαδοποίησης που στηρίζονται σε συναρτήσεις πυρήνα (kernel functions) και συγκεκριμένα στον αλγόριθμο Kernel k-Means που αποτελεί επέκταση του ευρέως γνωστού k-Means και ο οποίος χάρη στη συνάρτηση πυρήνα κατορθώνει να εντοπίζει μη γραμμικά διαχωρίσιμες ομάδες. Ωστόσο και αυτός πάσχει από το πρόβλημα της αρχικοποίησης. Για να το ξεπεράσουμε προτείνουμε έναν νέο αλγόριθμο τον Global Kernel k-Means που αποτελεί μία ντετερμινιστική και αυξητική προσέγγιση στο πρόβλημα της ομαδοποίησης με πυρήνες. Η μέθοδος αυτή προσθέτει μία ομάδα σε κάθε στάδιο μέσω μίας καθολικής μεθόδου αναζήτησης η οποία αποτελείται από πολλαπλές εκτελέσεις του Kernel k-Means από κατάλληλες αρχικοποιήσεις. Ο αλγόριθμος αυτός δεν εξαρτάται από την αρχικοποίηση, εντοπίζει μη γραμμικά διαχωρίσιμες ομάδες και βρίσκει σχεδόν βέλτιστες λύσεις αποφεύγοντας άσχημα τοπικά ελάχιστα. Επιπλέον προτείνουμε και δύο τροποποιήσεις οι οποίες μειώνουν το υπολογιστικό κόστος χωρίς ωστόσο να επηρεάζουν σημαντικά την ποιότητα των λύσεων σε πολλές περιπτώσεις.

Στηριζόμενοι στην παραλλαγή του Kernel k-Means με βάρη παρουσιάζουμε πως μπορούμε να προσαρμόσουμε τον Global Kernel k-Means και τις δύο τροποποιήσεις του ώστε να δέχονται βάρη. Η χρήση βαρών είναι σημαντική καθώς μπορεί να αποδειχθεί μία ισοδυναμία μεταξύ των μεθόδων αυτών και διάφορων κριτηρίων που εφαρμόζονται για τον διαμερισμό γράφων.

Οι προτεινόμενες μέθοδοι δοκιμάστηκαν σε διάφορα σύνολα δεδομένων όπως τεχνητά δεδομένα, MRI εικόνες, εικόνες προσώπων, χειρόγραφα ψηφία και γράφους και όπως προκύπτει υπερτερούν του Kernel k-Means με πολλαπλές εκκινήσεις.

Chapter 1

Introduction

As an ever increasing amount of information becomes available people try to find ways of storing, interpreting and handling this information. Usually this information is stored or represented as data. One of the vital means in dealing with these data is to *classify* or *group* them into a set of categories or clusters. In this way we get a condensed representation of the data that makes their interpretation a lot easier as similarities and differences as well as hidden structures in the data are exposed. In classification, which belongs to supervised learning, the task is to classify a data vector $\mathbf{x} \in \Re^d$ to one of a finite set of discrete class labels. In order to do this a classifier is built which maps input data to class labels through a mathematical function $y(\mathbf{x}, \mathbf{w})$ where $\mathbf{w}$ is a vector of adjustable parameters. Determining the values of these parameters is done through an inductive learning algorithm which usually minimizes an empirical risk function on a finite dataset $\mathcal{X} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_n \in \Re^d$ and y_n is the class of data vector $\mathbf{x}_n$.

Clustering, in which this work focuses on, belongs to unsupervised learning. The dataset does not contain data labels and is of the form $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ where $\mathbf{x}_n \in \Re^d$. The goal of clustering is to partition the dataset into a finite set of "natural", hidden structures. Exposing these hidden structures provides humans with meaningful insights from the original data which can be interpreted by experts in the field of the problem considered. That is why clustering has found application in a variety of fields such as machine learning, pattern recognition, image segmentation, spatial database analysis, life and medical sciences, economics and many more. Usually a measure of similarity or

dissimilarity, such as Euclidean distance, is used to describe the relations among data and clusters are created through a clustering algorithm which seeks to group data points in a way that patterns in the same cluster are similar to each other and dissimilar to those of other clusters. The algorithm usually identifies the clusters by optimizing a clustering criterion. The choice of proximity measure and clustering algorithm has a huge influence on the resulting clusters and different choices can lead to different number of clusters, different cluster shapes etc. Note that clustering is a subjective process in nature, as there is no absolute criterion on how data points should be grouped together, in contrast to classification where the class of each dataset point is available. Even the number of clusters is not known in advance. Thus, evaluating the clustering result is difficult and whether a solution is good or bad depends on the application under consideration.

In the remainder of this chapter we will describe the main types of clustering, some well known clustering algorithms and we will present a number of clustering methods that are based on the use of the proximity matrix. A proximity matrix $P \in \Re^{N \times N}$ contains the pairwise proximity (similarity or distance) of dataset points. A method of this type is the proposed Global Kernel k-Means algorithm studied in the following chapters. Finally, the purpose and the structure of this work are discussed.

1.1 Types of Clustering

There are two main types of clustering: *partitional* and *hierarchical*. Partitional clustering can be further divided into *hard* and *fuzzy*. Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_n \in \Re^d$, in hard partitional clustering we aim to divide it into a number, say M, of non-overlapping subsets (clusters) $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$, such that each data point is in exactly one subset. The subsets have the following properties:

1) $\mathcal{C}_i \neq \emptyset, \ i = 1, \ldots, M$

2) $\bigcup_{i=1}^{M} \mathcal{C}_i = \mathcal{X}$

3) $\mathcal{C}_i \cap \mathcal{C}_j = \emptyset, \ i, j = 1, \ldots, M$ and $i \neq j$

In fuzzy clustering each data point belongs to all clusters with a degree of membership $u_{ij} \in [0, 1]$ which represents the membership of point $\mathbf{x}_j$ in cluster $\mathcal{C}_i$. In other words, clusters are treated as fuzzy sets. This approach avoids the arbitrariness of hard clustering of assigning a data point to one cluster when it may be close to several. In practice, a fuzzy clustering can be converted to a hard clustering by assigning each point to the cluster in which its membership degree is highest. The membership degrees must satisfy the following two constraints:

1) $\sum_{i=1}^{M} u_{ij} = 1, \ \forall j$

2) $\sum_{j=1}^{N} u_{ij} < N, \ \forall i$

Input: Proximity matrix P
Output: Hierarchical clusters

// The algorithm starts with N singleton clusters
1: **repeat**
2: Merge the closest two clusters
3: Update the proximity matrix to reflect the proximity between the new cluster and the other clusters
4: **until** Only one cluster remains

Algorithm 1: Basic Agglomerative Clustering.

If we permit clusters to have subclusters then we obtain hierarchical clustering which is a set of nested clusters organized as a tree. Each node (cluster) in the tree, except for the leaf nodes, is the union of its children (subclusters) and the root of the tree is the cluster containing the whole dataset $\mathcal{X}$. Often the leaves of the tree are singleton clusters of individual data points. This approach is very useful when hierarchical relations exist in the dataset, like data from evolutionary research on different species of organisms. Finally, note that hierarchical clustering can be viewed as a sequence of hard partitional clusterings.

1.2 Basic Clustering Methods

In this section we review some of the most popular and widely used clustering algorithms that perform either hierarchical or partitional clustering. A comprehensive survey on clustering algorithms can be found in [26].

1.2.1 Hierarchical Algorithms

Hierarchical clustering algorithms organize data into a hierarchical structure according to the proximity matrix. There are two basic approaches for generating a hierarchical clustering, the *agglomerative* and the *divisive* methods. Agglomerative methods follow a bottom-up analysis and start with N singleton clusters. At each step the closest[1] pair of clusters is merged until all dataset points belong to one and only cluster. Divisive methods proceed in the opposite way and follow a top-down analysis. In the beginning the entire dataset belongs to a single cluster. At each step a cluster is split until singleton clusters of individual points remain. Agglomerative techniques are by far the most common in practice and on the remainder of this section we will focus on these methods. Note that most agglomerative algorithms are variations of the technique described in Algorithm 1.

Algorithm 1 contains the notion of cluster proximity. Based on the different definitions of cluster proximity there are many agglomerative clustering algorithms. The most

[1]If proximity is defined as similarity then closest clusters are the most similar ones while if proximity is defined as distance then closest clusters are the least distant ones.

popular ones are:

- *Single Link*: The proximity between two clusters is defined as the maximum similarity (or minimum distance) between any pair of points in the two different clusters. In mathematical terms, assuming that proximity is defined as similarity, the similarity between two clusters is:

$$s(\mathcal{C}_k, \mathcal{C}_l) = \max_{\mathbf{x}_i \in \mathcal{C}_k,\ \mathbf{x}_j \in \mathcal{C}_l} s(\mathbf{x}_i, \mathbf{x}_j) \tag{1.1}$$

- *Complete Link*: The proximity between two clusters is defined as the minimum similarity (or maximum distance) between any pair of points in the two different clusters. In mathematical terms, assuming that proximity is defined as similarity, the similarity between two clusters is:

$$s(\mathcal{C}_k, \mathcal{C}_l) = \min_{\mathbf{x}_i \in \mathcal{C}_k,\ \mathbf{x}_j \in \mathcal{C}_l} s(\mathbf{x}_i, \mathbf{x}_j) \tag{1.2}$$

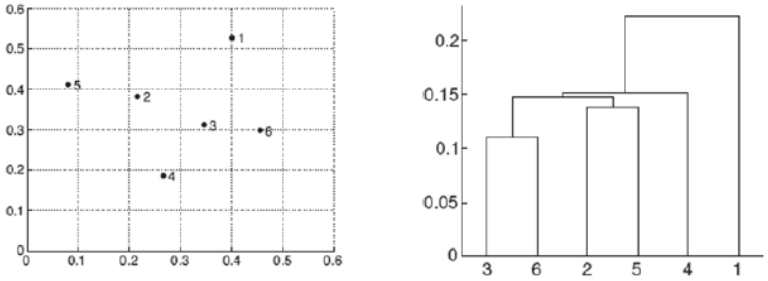
- *Group Average*: The proximity between two clusters is defined as the average pairwise proximity among all pairs of points in the different clusters. It is an intermediate approach between the single link and complete link approaches. In mathematical terms, assuming that proximity is defined as similarity, the similarity between two clusters is:

$$s(\mathcal{C}_k, \mathcal{C}_l) = \frac{\sum_{\mathbf{x}_i \in \mathcal{C}_k,\ \mathbf{x}_j \in \mathcal{C}_l} s(\mathbf{x}_i, \mathbf{x}_j)}{|\mathcal{C}_k|\,|\mathcal{C}_l|} \tag{1.3}$$

Several other agglomerative clustering algorithms, including centroid linkage, median linkage and Ward's method, can be constructed by choosing different cluster proximity measures. Single link, complete link and group average consider all points of a pair of clusters when calculating their proximity and are also called graph methods. The others are called geometric methods since they use geometric centers to represent the clusters and define cluster proximity.

Some key issues in hierarchical clustering are the following: most classical hierarchical algorithms lack robustness i.e. they are sensitive to noise and outliers. Also merging decisions are final hence, if at some stage a decision to merge two clusters is made it cannot be undone at a later stage. This is a drawback, as correcting previous misclassifications becomes impossible. Most of these methods have a computational time of $O(N^2 \log N)$ which limits their application to large scale datasets. In order to tackle some of these problems in recent years some new hierarchical clustering techniques have been proposed such as CURE [12] and BIRCH [30].

The result of hierarchical clustering can be represented using a tree-like diagram called *dendrogram* which displays the cluster-subcluster relationships as well as the order in which the clusters were merged or split for the agglomerative and divisive approaches respectively. An example of a dendrogram produced with the single link technique is shown in Figure 1.1. Note that the height at which two clusters are merged in this dendrogram reflects the distance of the two clusters.



	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Figure 1.1: Single link clustering example. The dataset points, the distance matrix containing pairwise Euclidean distances and the resulting dendrogram are shown.

1.2.2 k-Means

k-Means [19] is one of the oldest and most popular clustering algorithms. It is a hard partitional clustering algorithm, thus given a dataset $\mathcal{X}$, k-Means provides M disjoint clusters. Note that the number of clusters is given as input to the algorithm and does not change during its execution. Each cluster is represented by a centroid which is usually, but not always, the mean of the points of the cluster (called center in this case). In order to partition the dataset a clustering criterion is optimized. The most popular criterion is the clustering error defined as the sum of squared Euclidean distances between each data point $\mathbf{x}_n$ and the cluster centroid $\mathbf{m}_k$ of the cluster $\mathcal{C}_k$ that $\mathbf{x}_n$ belongs to:

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \|\mathbf{x}_n - \mathbf{m}_k\|^2 \tag{1.4}$$

where $I(Y) = 1$ if Y is true and 0 otherwise. Cluster centroid $\mathbf{m}_k$ in this case is defined as the mean of the points in cluster $\mathcal{C}_k$. k-Means optimizes this function by starting with M initial centroids, usually chosen randomly, and using an iterative procedure which alternates between assigning each data point to its closest centroid and recalculating the cluster centroids based on the new assignments. This procedure is repeated until no point changes cluster or equivalently until the centroids remain the same and is described in Algorithm 2.

In the above formulation of k-Means, the proximity between a point and a cluster is defined as the squared Euclidean distance which is a natural choice for points in Euclidean space. There are cases though where other proximity measures are more appropriate such as cosine similarity for document clustering. Indeed k-Means can be used to optimize other clustering criteria apart from clustering error. Some examples are shown in Table 1.1. The last entry on the table, Bregman divergence $d_\varphi()$, is actually a class of proximity measures that includes the squared Euclidean distance, Mahalanobis distance and cosine

Input: Dataset $\mathcal{X}$, Number of clusters M, Initial centroids $\mathbf{m}_1, \ldots, \mathbf{m}_M$
Output: Final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_M$, Final centroids $\mathbf{m}_1, \ldots, \mathbf{m}_M$, Clustering error E

1: **for all** points $\mathbf{x}_n \in \mathcal{X}$ **do**
2: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
3: Compute $\|\mathbf{x}_n - \mathbf{m}_k\|^2$
4: Find $c^*(\mathbf{x}_n) = \arg\min_k \left(\|\mathbf{x}_n - \mathbf{m}_k\|^2\right)$
5: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
6: Update cluster $\mathcal{C}_k = \{\mathbf{x}_n | c^*(\mathbf{x}_n) = k\}$
7: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
8: Update centroid $\mathbf{m}_k = \frac{\sum_{\mathbf{x}_n \in \mathcal{C}_k} \mathbf{x}_n}{|\mathcal{C}_k|}$
9: **if** converged **then**
10: **return** Final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_M$, final centroids $\mathbf{m}_1, \ldots, \mathbf{m}_M$ and E calculated using (1.4)
11: **else**
12: Go to step 1

Algorithm 2: k-Means.

similarity. Note that changes in the objective function result in centroids that are not always the mean of the cluster.

Table 1.1: Popular k-Means objective functions.

Proximity Function	**Centroid**	**Objective Function**
Manhattan distance	Median	$\sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \|\mathbf{x}_n - \mathbf{m}_k\|_1$ (minimize)
Squared Euclidean distance	Mean	$\sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \|\mathbf{x}_n - \mathbf{m}_k\|^2$ (minimize)
Cosine similarity	Mean	$\sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \cos(\mathbf{x}_n, \mathbf{m}_k)$ (maximize)
Bregman divergence	Mean	$\sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) d_\varphi(\mathbf{x}_n, \mathbf{m}_k)$ (minimize)

k-Means is a very simple and easily implemented algorithm with a computational complexity of $O(\tau MNd)$, where τ denotes the number of iterations until convergence, which make its application practical for large datasets. On the other hand, it suffers from some well known limitations. First of all, it locates local and not global optima of the objective function as the final solution depends on the centroids initialization. To avoid this problem usually the algorithm is run many times, with different initializations, and the solution with the best objective function value is kept as the final solution. Likas et al. [18] proposed the Global k-Means algorithm that does not depend on cluster initialization and locates near optimal solutions through a deterministic and incremental procedure. This algorithm is described in more detail in section 2.1. Another limitation is that only linearly separable clusters are identified and usually it is overcome by mapping the dataset points to a higher dimensional feature space through a nonlinear transformation

and applying k-Means in the feature space. Kernel k-Means is an algorithm that applies this strategy and is studied in section 2.2.

When the squared Euclidean distance is used as the proximity measure, outliers can influence the clustering result so it is a good tactic to identify and remove outliers from the dataset beforehand. Also k-Means has difficulty in detecting the "natural" clusters when they have non-spherical shapes or different sizes and density. Moreover, the dataset must be available in the form of vectors since calculation of the centroids is required. This prohibits the application of k-Means when only the proximity matrix is available. A workaround is the k-Medoids algorithm. Finally, the clustering criterion optimized must be based on a metric proximity measure. Despite all the above problems, k-Means is widely used, especially with the clustering error as the optimization criterion.

1.2.3 Fuzzy c-Means

Fuzzy c-Means [14] is one of the most popular fuzzy clustering algorithms. It is the fuzzy version of k-Means. As already mentioned, in fuzzy clustering data points belong to all clusters with a degree of membership rather to one cluster as in hard partitional clustering. This is particularly useful when boundaries among clusters are not well separated and ambiguous. Fuzzy c-Means works in a similar way to k-Means, represents each cluster with a centroid and identifies clusters by minimizing a fuzzy version of the clustering error given by:

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{n=1}^{N} \sum_{k=1}^{M} (u_{kn})^p \|\mathbf{x}_n - \mathbf{m}_k\|^2 \tag{1.5}$$

where u_{kn} is the membership degree of data point $\mathbf{x}_n$ in cluster $\mathcal{C}_k$ that satisfies the two conditions of section 1.1. The exponent p is called the fuzzification parameter and determines the influence of the membership degrees and has a value in the range $p \in [1, \infty)$.

The minimization of the above clustering objective is done in a similar fashion to k-Means. Starting with an initial set of M centroids, usually chosen randomly, the algorithm alternates between updating the membership degrees and recalculating the cluster centroids until the centroids remain the same. The update formulas are given by the following two equations:

$$u_{kn} = \left(\sum_{j=1}^{M} \left(\|\mathbf{x}_n - \mathbf{m}_k\| / \|\mathbf{x}_n - \mathbf{m}_j\| \right)^{\frac{2}{p-1}} \right)^{-1} \tag{1.6}$$

$$\mathbf{m}_k = \left(\sum_{n=1}^{N} (u_{kn})^p \mathbf{x}_n \right) \Bigg/ \left(\sum_{n=1}^{N} (u_{kn})^p \right) \tag{1.7}$$

As we can see, a cluster centroid is calculated as a weighted average where the contribution of each point is weighted by its membership degree. Also note that the closer a point is to a cluster centroid the higher the membership degree gets. The fuzzification parameter plays an important role in the above formulas. If p is chosen to be near 1, fuzzy c-Means

Input: Dataset $\mathcal{X}$, Number of clusters M, Initial centroids $\mathbf{m}_1, \ldots, \mathbf{m}_M$, Fuzzification p
Output: Membership degrees $u_{kn},\ k = 1, \ldots, M,\ n = 1, \ldots, N$, Fuzzy clustering error E

1: **for all** points $\mathbf{x}_n \in \mathcal{X}$ **do**
2: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
3: Compute $\|\mathbf{x}_n - \mathbf{m}_k\|^2$
4: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
5: Update membership degree u_{kn} using (1.6)
6: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
7: Update centroid $\mathbf{m}_k$ using (1.7)
8: **if** converged **then**
9: **return** Membership degrees $u_{kn},\ k = 1, \ldots, M,\ n = 1, \ldots, N$ and E calculated using (1.5)
10: **else**
11: Go to step 1

Algorithm 3: Fuzzy c-Means.

behaves like standard k-Means and the membership degree for the closest cluster goes to 1 while for the others goes to 0. On the other hand, when p goes to infinity, memberships tend to $1/M$ while all cluster centroids tend to the overall dataset centroid. Usually $p = 2$. The fuzzy c-Means procedure is described in Algorithm 3.

In general, this method has the same strengths and weaknesses as k-Means, although it is somewhat more computationally expensive. Numerous fuzzy c-Means variants have been proposed as a result of the intensive investigation on the proximity measures, the effect of the fuzzification parameter and improvements of its drawbacks, such as the extension which uses Minkowski distance (L_p norm) [13].

1.2.4 Mixture Model Clustering

In the probabilistic view data objects are assumed to have been generated as a result of a statistical process. A convenient way to describe the data is to find the statistical model that best fits the data, where the statistical model is described in terms of a distribution and a set of parameters for that distribution. Mixture models assume that data are generated according to several probability distributions, called *components*, and each of these distributions represents a cluster. Data in different clusters are generated by different distributions i.e. distributions from different density functions (e.g. multivariate Gaussian or t-distribution) or distributions from the same family but with different parameters. If the distributions are known, finding the clusters of a given dataset is equivalent to estimating the parameters of the distributions. Once again we assume that the number of clusters M is known. The mixture model probability distribution is expressed as:

$$p(\mathbf{x}; \boldsymbol{\theta}) = \sum_{k=1}^{M} q_k p_k(\mathbf{x}; \boldsymbol{\theta}_k) \tag{1.8}$$

where q_k is the prior probability of the k-th component, $p_k(\mathbf{x};\boldsymbol{\theta}_k)$ is the conditional probability distribution for this component and represents the k-th cluster, while $\boldsymbol{\theta}_k$ is the unknown parameters vector for the k-th distribution. Finally, $\boldsymbol{\theta} = (\{\boldsymbol{\theta}_1, q_1\}, \ldots, \{\boldsymbol{\theta}_M, q_M\})$ is the mixture model parameters vector. Prior probabilities must sum to one in order $p(\mathbf{x};\boldsymbol{\theta})$ to be a probability distribution, thus $\sum_{k=1}^{M} q_k = 1$.

As long as the parameter vector $\boldsymbol{\theta}$ is decided, the posterior probability of assigning a data point to cluster k can be easily calculated with Bayes's theorem as:

$$p(k|\mathbf{x};\boldsymbol{\theta}) = \frac{q_k p_k(\mathbf{x};\boldsymbol{\theta}_k)}{p(\mathbf{x};\boldsymbol{\theta})} \tag{1.9}$$

A fact that becomes clear from the above assignment step is that mixture models produce something like a fuzzy partition of the dataset as each point belongs to each cluster with a certain probability, something similar to the membership degrees of fuzzy c-Means. In order to estimate the parameters of the mixture model, maximum likelihood estimation is used, which considers as the best estimate the one that maximizes the probability of generating the whole dataset $\mathcal{X}$. Usually the logarithmic form of the likelihood is used, called log-likelihood, and for mixture models is:

$$L(\boldsymbol{\theta};\mathcal{X}) = \sum_{n=1}^{N} \log p(\mathbf{x}_n;\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{k=1}^{M} q_k p_k(\mathbf{x}_n;\boldsymbol{\theta}_k) \tag{1.10}$$

Maximization of the log-likelihood function is performed with an iterative method, the popular expectation maximization (EM) algorithm. This algorithm starts with an initial guess of the parameters and then alternates between the expectation (E) step, where the probability that each point belongs to each distribution (cluster) is calculated, and the maximization (M) step, where new estimates for the parameters are computed using the posterior probabilities until the parameters do not change. The components of a mixture model can be of any type of probability distribution, but in most cases multivariate Gaussian distributions are used, resulting in a Gaussian mixture model (GMM), due to their complete theory and analytical tractability. GMM parameters are the prior probability q_k, the mean $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$ of each component. Calculation of the GMM parameters using the EM algorithm is shown in Algorithm 4.

The major disadvantages of mixture model clustering are the EM algorithm's slow convergence rate and its sensitivity to the selection of initial parameters. Depending on the initialization it may converge to local and not global maxima of the log-likelihood function. This problem grows as the number of components and adjustable parameters increases. Also there is the problem of singularities which occurs when a component "collapses" onto a specific data point. On the other hand, mixture models are more general than k-Means and fuzzy c-Means. In fact k-Means is a special case of the EM algorithm for a GMM whose components are spherical Gaussian distributions with the same covariance matrix and different means. GMMs can find clusters with elliptical shapes unlike k-Means which is limited to spherical clusters. Finally, fitting data to a model is a good way to identify patterns in them.

Input: Dataset $\mathcal{X}$, Number of components M,
Initial parameters $\boldsymbol{\theta} = (\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, q_1\}, \ldots, \{\boldsymbol{\mu}_M, \boldsymbol{\Sigma}_M, q_M\})$
Output: Final parameters $\boldsymbol{\theta} = (\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, q_1\}, \ldots, \{\boldsymbol{\mu}_M, \boldsymbol{\Sigma}_M, q_M\})$

1: **for all** points $\mathbf{x}_n \in \mathcal{X}$ **do** // E-step
2: **for all** components $k = 1$ to M **do**
3: Compute $p(k|\mathbf{x}_n; \boldsymbol{\theta})$ using (1.9)
4: **for all** components $k = 1$ to M **do** // M-step
5: Update mean $\boldsymbol{\mu}_k = \frac{\sum_{n=1}^{N} p(k|\mathbf{x}_n;\boldsymbol{\theta})\mathbf{x}_n}{\sum_{n=1}^{N} p(k|\mathbf{x}_n;\boldsymbol{\theta})}$
6: Update covariance matrix $\boldsymbol{\Sigma}_k = \frac{\sum_{n=1}^{N} p(k|\mathbf{x}_n;\boldsymbol{\theta})(\mathbf{x}_n - \boldsymbol{\mu}_k)(\mathbf{x}_n - \boldsymbol{\mu}_k)^{\mathrm{T}}}{\sum_{n=1}^{N} p(k|\mathbf{x}_n;\boldsymbol{\theta})}$
7: Update prior probability $q_k = \frac{\sum_{n=1}^{N} p(k|\mathbf{x}_n;\boldsymbol{\theta})}{N}$
8: **if** converged **then**
9: **return** Final parameters $\boldsymbol{\theta} = (\{\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, q_1\}, \ldots, \{\boldsymbol{\mu}_M, \boldsymbol{\Sigma}_M, q_M\})$
10: **else**
11: Go to step 1

Algorithm 4: EM for Gaussian mixture models.

1.3 Proximity and Kernel Matrix Based Clustering

A *proximity* matrix $P \in \Re^{N \times N}$ contains the pairwise proximity (similarity or distance) $P_{ij} = prox(\mathbf{x}_i, \mathbf{x}_j)$ of all dataset points. If the proximity is distance it is called a *distance* matrix $D \in \Re^{N \times N}$ and $D_{ij} = d(\mathbf{x}_i, \mathbf{x}_j)$, while if it is similarity it is called a *similarity* matrix $S \in \Re^{N \times N}$ and $S_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$. Algorithms such as k-Means, fuzzy c-Means, GMMs and many others require as input the dataset points in the form of vectors. There are cases though where only the proximity matrix is available making the application of these methods impossible. This happens basically for two reasons. The pairwise proximities are extracted by the dataset owner and only these are published or data points may not be vectors at all. Consider for example graph clustering where graph vertices are the data and edge weights describe their proximity. In this problem there are no vectors and only a proximity matrix can be extracted. For these reasons clustering algorithms that can directly work on the proximity matrix have been developed and in this section we will review some of them. Also in this category of algorithms belong methods that take as input a *kernel* matrix $K \in \Re^{N \times N}$ where $K_{ij} = \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}_j)$. Function ϕ is a nonlinear transformation that maps data to a higher dimensional space. Usually the kernel matrix is produced without explicitly defining ϕ through a kernel function. Note that any positive semidefinite matrix can be thought as a kernel matrix. For more details on the kernel matrix see section 2.2. We focus on proximity and kernel matrix based clustering as a lot of interesting algorithms in this area have been lately developed including the proposed Global Kernel k-Means algorithm.

Input: Distance matrix D, Number of clusters M, Initial medoids $\mathbf{m}_1, \ldots, \mathbf{m}_M$
Output: Final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_M$

1: **for all** points $\mathbf{x}_n$ $n = 1, \ldots, N$ **do**
2: Find $c^*(\mathbf{x}_n) = \arg\min_k d(\mathbf{x}_n, \mathbf{m}_k)$
3: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
4: Update cluster $\mathcal{C}_k = \{\mathbf{x}_n | c^*(\mathbf{x}_n) = k\}$
5: **for all** clusters $\mathcal{C}_k$ $k = 1$ to M **do**
6: Update medoid $\mathbf{m}_k = \mathbf{x}_{n^*}, n^* = \arg\min_{\mathbf{x}_n \in \mathcal{C}_k} \sum_{\mathbf{x}_i \in \mathcal{C}_k} d(\mathbf{x}_i, \mathbf{x}_n)$
7: **if** converged **then**
8: **return** final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_M$
9: **else**
10: Go to step 1

Algorithm 5: k-Medoids.

1.3.1 k-Medoids

k-Medoids is closely related to k-Means and produces a hard partitioning of the dataset. Their main difference is that k-Medoids represents a cluster with a medoid instead of a centroid. While the centroid does not in general correspond to a dataset point, the medoid, by its definition, must be a dataset point. The use of the medoid instead of the centroid is what allows k-Medoids to work using the proximity matrix without requiring the dataset in vectorial form. One can think of the medoid as the most "central" data point of the cluster with respect to some proximity measure.

k-Medoids splits the dataset into a predefined number M of clusters by optimizing the following clustering criterion:

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) prox(\mathbf{x}_n, \mathbf{m}_k) \tag{1.11}$$

where $\mathbf{m}_k$ is the cluster medoid, $\mathbf{m}_k \in \mathcal{X}$ and $prox(\mathbf{x}_n, \mathbf{m}_k)$ is the proximity between data point $\mathbf{x}_n$ and medoid $\mathbf{m}_k$ given by the proximity matrix. If the proximity is similarity the above criterion is maximized, while if it is distance it is minimized. The optimization is done with an iterative procedure almost identical to that of k-Means. The algorithm starts by selecting an initial set of data points as medoids, usually randomly, and proceeds by alternating between assigning each data point to the closest medoid and discovering the best data point to serve as medoid for each cluster, until the medoids do not change. The closest medoid is found through a simple look up at the proximity matrix. The medoid of a cluster is a point that belongs to that cluster and is found through a discrete search over the cluster points. As medoid for the k-th cluster is selected the point $\mathbf{x}_{n^*}$, where $n^* = \arg\min_{\mathbf{x}_n \in \mathcal{C}_k} \sum_{\mathbf{x}_i \in \mathcal{C}_k} prox(\mathbf{x}_i, \mathbf{x}_n)$, if the proximity is distance or by the same formula but with $\arg\max$ in place of $\arg\min$ if proximity is similarity. This search can be performed using only the proximity matrix. The procedure is described in Algorithm 5 where the use of distance as the proximity is assumed.

The advantages of k-Medoids over k-Means are that it requires only the proximity matrix, the use of medoids makes cluster representatives more robust to outliers and that any similarity or distance measure can be used in the objective function as long as it can be readily evaluated. Thus k-Medoids can optimize all of the k-Means objectives and many more. There are also a number of drawbacks such as the higher computational complexity. k-Medoids requires $O(\tau(N^2 + MN))$ operations which is bad for large datasets. This complexity is a result of the discrete search required to identify the medoids. Also in some cases the medoid may not be a good representative as shown in Figure 1.2. Finally, only local optima of the clustering criterion are identified as the solution depends on the initial medoids.

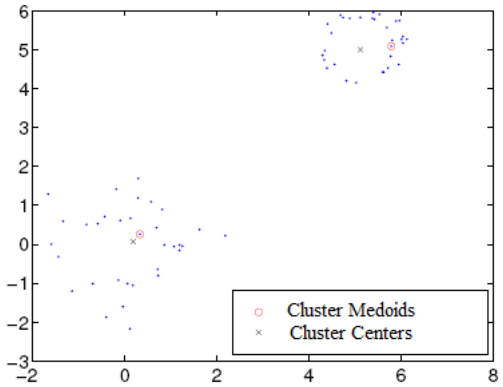


Figure 1.2: Medoids and centers of two clusters. For the ring-like cluster the medoid is not a good representative as it is far away form the cluster center.

1.3.2 Spectral Clustering

Spectral clustering is a relatively new approach to clustering that produces a hard partitioning of the dataset using the eigenvectors of a matrix derived from the data. More specifically, a matrix containing the pairwise similarities, called *affinity* matrix in this case, is used and the eigenvectors of this matrix or a matrix derived from the affinity matrix are calculated. The eigenvectors are then processed to obtain a clustering of the dataset. The affinity matrix, $A \in \Re^{N \times N}$ and $A_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$, is either constructed from the data points using a measure of similarity or it is provided by a third party and is directly used.

A number of different spectral algorithms appear in the literature which basically differ on the matrix used to calculate the eigenvectors and the way the eigenvectors are processed to obtain the final partitioning. Some of them are summarized in [25]. Here we present the algorithm proposed by Ng et al. [20] which is described in Algorithm 6[2]. Other spectral methods follow a similar approach. It is obvious that this algorithm does not require as input the dataset in vectorial form. If the dataset is available though Ng et al. proposed computing the affinity matrix as $A_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma^2)$ for $i \neq j$

[2]Largest eigenvector is the one corresponding to the largest eigenvalue.

Input: Affinity matrix A, Number of clusters M
Output: Final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_M$

1: Define the diagonal matrix D where $D_{ii} = \sum_{j=1}^{N} A_{ij}$
2: Construct the matrix $L = D^{-1/2} A D^{-1/2}$
3: Calculate $e_1, \ldots, e_M$ the M largest eigenvectors of L and form the matrix $E = [e_1, \ldots, e_M] \in \Re^{N \times M}$
4: Normalize each of E's rows to unit length and construct matrix Y thus $Y_{ij} = E_{ij} \Big/ \left(\sum_{l=1}^{M} E_{il}^2\right)^{1/2}$
5: Treat each row of Y as a point in $\Re^M$ and cluster them into M clusters using k-Means (the version that optimizes clustering error)
6: Assign the original point $\mathbf{x}_n$ to cluster $\mathcal{C}_k$ only if the n-th row of matrix Y was assigned to the k-th cluster

Algorithm 6: Spectral clustering with the Ng et al. algorithm.

and $A_{ii} = 0$. The parameter σ controls how rapidly the affinity falls off with the distance. One may wonder since in step 5 we apply k-Means on the eigenvectors why not apply it directly to the data. The answer is that mapping the points in $\Re^M$ using eigenvectors can lead to tight clusters that can be identified by k-Means. An example is shown in Figures 1.3 - 1.4, where the two rings cannot be identified directly by k-Means, since they are not linearly separable, but when they are mapped to $\Re^2$, through the eigenvectors, this is possible.

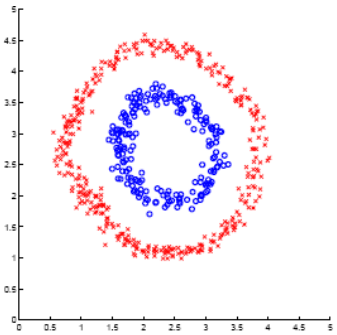


Figure 1.3: Two rings dataset. The rings cannot be separated by k-Means in the original space.

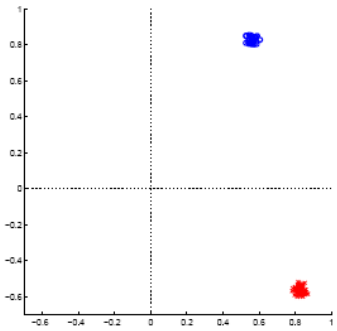


Figure 1.4: The two rings mapped in $\Re^2$ with the use of eigenvectors. Now the rings can be separated with k-Means.

To understand how this algorithm works, consider the "ideal" case in which data points belonging to different clusters are infinitely far apart, hence their affinity is zero. This results in an affinity matrix that is block diagonal and thus matrix L is also block diagonal. The eigenvectors and eigenvalues of a block diagonal matrix are the union of the eigenvalues and eigenvectors of its blocks (the latter padded appropriately with zeros). From linear algebra it is known that each block has an eigenvalue equal to 1 and the next eigenvalue is strictly less than 1. Thus taking as many largest eigenvectors as the number of blocks from matrix L, results in taking the largest eigenvector of each block padded

appropriately with zeros. Then matrix E is of the form:

$$E = \begin{bmatrix} e_1^{(1)} & \vec{0} & \cdots & \vec{0} & \vec{0} \\ \vec{0} & e_1^{(2)} & \cdots & \vec{0} & \vec{0} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \vec{0} & \vec{0} & \cdots & e_1^{(M-1)} & \vec{0} \\ \vec{0} & \vec{0} & \cdots & \vec{0} & e_1^{(M)} \end{bmatrix} \in \Re^{N \times M}$$

where $e_1^{(k)}$ is the largest eigenvector of the k-th block. Clustering the rows of this matrix (or the normalized rows of matrix Y) is straightforward as those with non zero value in the first dimension belong to the first cluster, those with non zero value in the second dimension belong to the second cluster etc. Note that this clustering corresponds to the true clustering of the original data points. This is the intuition in which all spectral clustering methods are based. Obviously in real problems the affinity between points in different clusters will not be zero and the largest eigenvectors will not define so clearly a partition but with some processing we expect to approach the true clusters.

The main advantage of spectral methods is that they do not depend on initializations. Only the step that derives the clusters from the eigenvectors may require initialization (k-Means in Algorithm 6) but this will not change the final solution considerably if the eigenvectors strongly indicate a partitioning of the dataset. Also there is no limitation on the form of the affinity matrix which can contain non metric similarities. Spectral methods have been employed to numerous problems with satisfactory results. They are particularly popular for graph partitioning as many graph cut criteria can be optimized by taking the eigenvectors of a matrix derived from the graph affinity matrix. Spectral methods compute a globally optimum solution of a *relaxation* of the graph problem. The main disadvantage is the high computational complexity caused by the need to compute the eigenvalues and eigenvectors of an $N \times N$ matrix. Eigenvalue decomposition costs $O(N^3)$ which is very high for large datasets. As only the M largest eigenvectors are required special mathematical packages are used that approximate them without computing all eigenvectors. Still though this operation costs a lot.

1.3.3 Affinity Propagation

Affinity propagation was recently proposed by Frey et al. [11] and is a hard partitioning algorithm that clusters data by identifying representative exemplars. An exemplar is an actual dataset point that represents a cluster, similar to a medoid. Unlike most algorithms which search for a specified number of clusters and start with a predefined set of exemplars that is iteratively refined, affinity propagation initially considers all dataset points as possible exemplars and the number of clusters is defined during the learning process. Affinity propagation views each data point as a node in a network that transmits real valued messages along the edges of the network until a good set of exemplars and corresponding clusters emerges.

The input to the algorithm is a similarity matrix $S \in \Re^{N \times N}$ where the similarity $s(i,k)$ indicates how well data point $\mathbf{x}_k$ is suited to be an exemplar for data point $\mathbf{x}_i$. Self-similarities $s(k,k)$ are called *preferences* and the higher this value is, the more likely is that data point $\mathbf{x}_k$ will be chosen as an exemplar. Thus the number of identified exemplars (clusters) is influenced by the values of the preferences and by the message passing procedure. Note that preferences are independent from the other similarities and are not calculated in the same way, since they do not represent assignment similarities. If all dataset points are equally suitable for exemplars at the beginning, then preferences should be set to a common value. For example, we could set preferences equal to the median of the similarities, resulting in a moderate number of clusters, or equal to the minimum similarity, resulting in a small number of clusters.

Affinity propagation produces a partitioning of the dataset by minimizing the following energy function:

$$E(c_1, \ldots, c_N) = -\sum_{i=1}^{N} s(i, c_i) \tag{1.12}$$

where c_i is the index of the data point selected as the exemplar that represents the cluster that $\mathbf{x}_i$ belongs to. Thus $s(i, c_i)$ is the similarity between $\mathbf{x}_i$ and its cluster exemplar. A constraint must be satisfied to obtain a meaningful clustering: for all i, if there exists a $j \neq i$ and $c_j = i$ then $c_i = i$. This means that if point $\mathbf{x}_i$ is selected as an exemplar by another point then $\mathbf{x}_i$ must select itself as its exemplar. The minimization of the above energy function is done by exchanging messages between data points. There are two kinds of messages, the *responsibility* and the *availability*. Responsibility $r(i,k)$ sent from data point $\mathbf{x}_i$ to candidate exemplar $\mathbf{x}_k$ reflects the accumulated evidence about how well suited point $\mathbf{x}_k$ is to serve as the exemplar of point $\mathbf{x}_i$ taking into account other potential exemplars for $\mathbf{x}_i$. Availability $a(i,k)$ sent from candidate exemplar $\mathbf{x}_k$ to point $\mathbf{x}_i$ reflects the accumulated evidence about how appropriate it would be for point $\mathbf{x}_i$ to choose $\mathbf{x}_k$ as its exemplar, taking into account the support from other points that $\mathbf{x}_k$ should be an exemplar.

Affinity propagation starts by initializing all availabilities to zero and then alternates between calculating the responsibilities and the availabilities. The responsibilities are defined as:

$$r(i,k) = s(i,k) - \max_{j:j \neq k} \{a(i,j) + s(i,j)\} \tag{1.13}$$

For $k = i$ responsibility $r(k,k)$ reflects accumulated evidence that $\mathbf{x}_k$ is an exemplar based on its preference and how ill-suited it is to be assigned to another exemplar. This quantity is also called *self-responsibility*. The availability is given by the following equation:

$$a(i,k) = \min \left\{ 0,\ r(k,k) + \sum_{j:j \notin \{i,k\}} \max \{0,\ r(j,k)\} \right\} \tag{1.14}$$

Note that only the positive incoming responsibilities are added as a good exemplar needs to explain some data points well (positive responsibilities) regardless of how poorly it explains other data points (negative responsibilities). The self-availability $a(k,k)$ is updated

differently as:

$$a(k,k) = \sum_{j:j\neq k} \max\{0,\ r(j,k)\} \tag{1.15}$$

This message reflects accumulated evidence that $\mathbf{x}_k$ is an exemplar based on the positive responsibilities sent to candidate exemplar $\mathbf{x}_k$ from other points. At any point during execution we can combine availabilities and responsibilities to identify exemplars. The assignment of a point $\mathbf{x}_i$ is done by finding the value of k that maximizes $a(i,k)+r(i,k)$. If $k = i$ then $\mathbf{x}_i$ is an exemplar, otherwise it selects $\mathbf{x}_k$ as its exemplar. Data points that select the same exemplar belong to the same cluster. The algorithm converges after a predefined number of iterations or if the exemplars do not change for a number of consecutive iterations.

The practical implementation of the algorithm differs slightly from the above theoretical presentation. The update formulas for the availabilities and the responsibilities are damped to avoid numerical oscillations and also the exemplars are identified as those $\mathbf{x}_k$ for which $a(k,k)+r(k,k) > 0$. The rest of the dataset points are assigned to the most similar exemplar. The assignment formula described above is not used. This implementation is shown in Algorithm 7.

Affinity propagation was applied on a number of machine learning tasks including clustering face images, finding genes using microarray data, document summarization and airline routing. The method was compared to the k-Centers algorithm, GMMs and hierarchical agglomerative algorithms and when the problem had many clusters it achieved better results in less time. For more details see [10, 11]. It is obvious that affinity propagation can handle non vectorial data as long as the similarity matrix is provided. One of its biggest advantages is that it does not require the similarities to be metric. Only few algorithms work with non metric proximity measures, e.g. spectral methods. This allows affinity propagation to optimize a variety of clustering criteria. Many experiments in [10, 11] use non metric similarities and the results are very promising. For Euclidean data if we set $s(i,k) = -\|\mathbf{x}_i - \mathbf{x}_k\|^2$ then equation (1.12) becomes analogous to the clustering error. A second advantage is that the number of clusters is not defined in advance but is discovered during the learning process. Of course the problem of finding the optimal number of clusters is not completely overcome as the preference parameter considerably affects the final number of clusters. Since all points are considered as possible exemplars this method avoids poor minima of the energy function that other algorithms such as k-Means, k-Medoids and mixture models suffer from due to bad initializations. Overall, it is a promising method but relatively new, so work must be done to discover its pros and cons in depth.

1.3.4 Other Algorithms

We briefly discuss here two other algorithms that fall in the category of proximity and kernel matrix based clustering and are further analyzed on the following chapters. Kernel k-Means [22] is an extension of the k-Means algorithm that avoids the problem of linearly

Input: Similarity matrix S, Damping factor $\lambda \in [0,1]$
Output: Assignment of points to exemplars $c_1, \ldots, c_N$

1: $t = 1$, $Exem = \emptyset$
2: **for all** points $\mathbf{x}_i$ $i = 1, \ldots, N$ **do**
3: **for all** points $\mathbf{x}_k$ $k = 1, \ldots, N$ **do**
4: Initialize availabilities $a^{(0)}(i,k) = 0$
5: Initialize responsibilities $r^{(0)}(i,k) = 0$
6: **for all** points $\mathbf{x}_i$ $i = 1, \ldots, N$ **do**
7: **for all** points $\mathbf{x}_k$ $k = 1, \ldots, N$ **do**
8: Calculate responsibility $r^{(t)}(i,k) = (1-\lambda)\left(s(i,k) - \max_{j:j\neq k}\left\{a^{(t-1)}(i,j) + s(i,j)\right\}\right) + \lambda r^{(t-1)}(i,k)$
9: **for all** points $\mathbf{x}_i$ $i = 1, \ldots, N$ **do**
10: **for all** points $\mathbf{x}_k$ $k = 1, \ldots, N$ $k \neq i$ **do**
11: Calculate availabilities $a^{(t)}(i,k) = (1-\lambda)\left(\min\left\{0,\ r^{(t)}(k,k) + \sum_{j:j\notin\{i,k\}} \max\left\{0,\ r^{(t)}(j,k)\right\}\right\}\right) + \lambda a^{(t-1)}(i,k)$
12: **for all** points $\mathbf{x}_k$ $k = 1, \ldots, N$ **do**
13: Calculate self-availabilities $a^{(t)}(k,k) = (1-\lambda)\left(\sum_{j:j\neq k} \max\left\{0,\ r^{(t)}(j,k)\right\}\right) + \lambda a^{(t-1)}(k,k)$
14: **if** converged **then**
15: **for all** points $\mathbf{x}_k$ $k = 1, \ldots, N$ **do**
16: **if** $a^{(t)}(k,k) + r^{(t)}(k,k) > 0$ **then** // point is an exemplar
17: $c_k = k$, $Exem = Exem \cup k$
18: **for all** points $\mathbf{x}_i$ $i = 1, \ldots, N$ $i \notin Exem$ **do**
19: $c_i = \arg\max_{k\in Exem} s(i,k)$
20: **return** Assignment of points to exemplars $c_1, \ldots, c_N$
21: **else**
22: $t = t + 1$ Go to step 6

Algorithm 7: Affinity propagation.

separable clusters. The dataset points are mapped to a higher dimensional feature space through a nonlinear transformation ϕ and k-Means is applied on the feature space. Kernel k-Means optimizes the clustering error but in feature space instead of the input space and requires as input a kernel matrix, where $K_{ij} = \phi(\mathbf{x}_i)^{\mathrm{T}}\phi(\mathbf{x}_j)$, and not the dataset. As already mentioned, the entries of the kernel matrix are usually calculated through a kernel function without explicitly defining transformation ϕ. The algorithm is described in detail in section 2.2.

Convex mixture model clustering proposed by Lashkari et al. [16] is a new method that tries to identify exemplars on a dataset through mixture model fitting and considers all points as possible exemplars. The proposed mixture model has some substantial differences to the standard mixture model presented in section 1.2.4. First of all, it has as many components as the number of data points. Second, the components are exponential family distributions but have no free parameters to optimize. The expected value of the n-th component's exponential distribution is set equal to $\mathbf{x}_n$. Actually, the only adjustable parameters of the mixture model are the prior probabilities. Also there is a temperature-

like parameter β that controls the number of identified clusters. This formulation of the mixture model leads to a convex minimization problem whose globally optimal solution can always be found with a simple iterative method. This method requires as input a similarity matrix of the data points where $S_{ij} = \exp(-\beta d(\mathbf{x}_i, \mathbf{x}_j))$ and $d(\mathbf{x}_i, \mathbf{x}_j)$ is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ and is derived from the distance measure corresponding to the exponential distribution used in the mixture model. The authors claim though that their method can be applied to arbitrary distance matrices as well, where distances can even be asymmetric. Convexity helps to avoid poor solutions found with the EM algorithm for standard mixture models due to bad initialization and the large number of parameters to be optimized. On the other hand, reducing the number of adjustable parameters can limit the flexibility of the mixture model. This method is further discussed in section 3.2.2.

Finally, hierarchical clustering algorithms, already presented in section 1.2.1, also take as input a proximity matrix and do not require the dataset to be available.

1.4 Purpose and Structure of the Thesis

The above discussion has provided some insights into what clustering is and the different types of methods used for this task. Also, the main advantages and disadvantages of each algorithm were analyzed and it becomes clear that a "best of all" algorithm does not exist. Having that in mind, in this work we develop a new kernel-based method, called Global Kernel k-Means, that tries to overcome the two basic problems of k-Means, its dependence on initialization and the inability to identify nonlinearly separable clusters. This method combines the ideas of the Global k-Means [18] and Kernel k-Means [22] algorithms to achieve that goal. Knowing that computation time has an important role on the applicability of a method, two variants are proposed to speed up the execution. One of them follows a similar approach to the Fast Global k-Means variant of Global k-Means, while the other is an integration of the convex mixture model clustering algorithm [16] with Global Kernel k-Means. Also a weighted version of the algorithms is presented and its close relation with graph cuts is discussed. Our main priority is to evaluate the performance of the developed algorithms and determine if in practice they overcome the limitations of k-Means. Furthermore, we want to compare the two variants with the original algorithm and see at what cost, in terms of clustering solution quality, the computation savings are achieved. The applicability of the proposed methods on different clustering problems is another concern and for this reason we perform experiments with many datasets and report the obtained results.

The rest of this work is organized as follows: chapter 2 presents the Global k-Means and Kernel k-Means algorithms on which the proposed algorithm is based. The Global Kernel k-Means algorithm is described in chapter 3 together with two speeding up schemes, while the weighted versions are presented in chapter 4. The performance of the algorithms is evaluated in the following chapter where they are compared to Kernel k-Means with multiple restarts on many clustering problems. Finally, chapter 6 concludes this work.

Chapter 2

Preliminaries

This chapter focuses on the Global k-Means and Kernel k-Means algorithms. These two methods are of great significance for this work, as the proposed Global Kernel k-Means algorithm is derived based on the ideas of these methods. This is the reason we devote a separate chapter for these algorithms instead of presenting them in the introduction.

2.1 k-Means and Global k-Means

Suppose we are given a dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_n \in \Re^d$ and we aim to partition this dataset into M disjoint clusters $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$. The k-Means algorithm, presented in section 1.2.2, finds local optimal solutions with respect to the clustering error defined as the sum of squared Euclidean distances between each data point $\mathbf{x}_n$ and the cluster center $\mathbf{m}_k$ that $\mathbf{x}_n$ belongs to. Analytically the clustering error is given by:

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \|\mathbf{x}_n - \mathbf{m}_k\|^2 \tag{2.1}$$

where $I(Y) = 1$ if Y is true and 0 otherwise.

The two main disadvantages of the k-Means algorithm are first the dependence of the final solution on the initial position of the cluster centers and second that clusters must be linearly separable. To deal with the initialization problem the Global k-Means algorithm has been proposed [18], an incremental-deterministic algorithm that employs the k-Means algorithm as a local search procedure. This algorithm obtains near optimal solutions in terms of clustering error.

In order to solve the M-clustering problem using Global k-Means we proceed as follows. We begin by solving the 1-clustering problem using k-Means. The optimal solution to

Input: Dataset $\mathcal{X}$, Number of clusters M
Output: Final clustering of the points $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$

// There is no need to solve for 1 cluster as the solution is trivial and optimal. $\mathcal{C}_1 = \mathcal{X}$
1: Calculate dataset centroid $\mathbf{m}_1^* = \frac{\sum_{n=1}^{N} \mathbf{x}_n}{N}$
2: **for all** k-clustering problems $k = 2$ to M **do**
3: **for all** points $\mathbf{x}_n \in \mathcal{X}$ **do**
4: Run k-Means with: **input** $\left(\mathcal{X}, k, \mathbf{m}_1^*, \ldots, \mathbf{m}_{k-1}^*, \mathbf{m}_k = \mathbf{x}_n\right)$
output $(\mathcal{C}_1^n, \ldots, \mathcal{C}_k^n, \mathbf{m}_1^n, \ldots, \mathbf{m}_k^n, E_k^n)$
5: Find $E_k^* = \min_n(E_k^n)$ and set $(\mathcal{C}_1^*, \ldots, \mathcal{C}_k^*)$, $(\mathbf{m}_1^*, \ldots, \mathbf{m}_k^*)$ to the partitioning and centers respectively corresponding to E_k^*
// This is the solution with k clusters
6: **return** $\mathcal{C}_1 = \mathcal{C}_1^*, \ldots, \mathcal{C}_M = \mathcal{C}_M^*$ as output of the algorithm

Algorithm 8: Global k-Means.

this problem is known and the cluster center corresponds to the dataset centroid. Then we solve the 2-clustering problem. We run k-Means N times, each time starting with the following initial cluster centers: one cluster center is always placed at the position resulting from the 1-clustering problem and the other during the n-th run is initially placed at data point $\mathbf{x}_n$. The solution with the lowest clustering error is kept as the solution of the 2-clustering problem. In general for the k-clustering problem let $\left(\mathbf{m}_1^*, \ldots, \mathbf{m}_{k-1}^*\right)$ denote the solution to the $(k-1)$-clustering problem. We perform N executions of the k-Means algorithm, with $\left(\mathbf{m}_1^*, \ldots, \mathbf{m}_{k-1}^*, \mathbf{x}_n\right)$ as initial cluster centers for the n-th run, and keep the one resulting in the lowest clustering error. The above procedure is repeated until $k = M$. The pseudo code is shown in Algorithm 8.

It is obvious that the above algorithm does not suffer from the initialization of the cluster centers problem and computes a clustering of the data points in a deterministic way. Also it provides all intermediate solutions with $1, \ldots, M$ clusters when solving the M-clustering problem without additional cost. The experiments performed in [18] verify that Global k-Means is better than k-Means with multiple restarts. A drawback of Global k-Means is that it is computationally heavy as it requires running the k-Means algorithm MN times. To speed up execution two variants of the Global k-Means algorithm are proposed in [18] that do not considerably degrade the performance of the algorithm. Another variant was recently developed by Bagirov [2].

2.2 Kernel k-Means

Kernel k-Means [22] is a generalization of the standard k-Means algorithm where data points are mapped from input space to a higher dimensional feature space through a nonlinear transformation ϕ and then k-Means is applied in the feature space. This results in linear separators in feature space which correspond to nonlinear separators in input space. Thus Kernel k-Means avoids the limitation of linearly separable clusters in input

Input: Kernel matrix K, Number of clusters k, Initial clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$
Output: Final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$, Clustering error E

1: **for all** points $\mathbf{x}_n$ $n = 1, \ldots, N$ **do**
2: **for all** clusters $\mathcal{C}_i$ $i = 1$ to k **do**
3: Compute $\|\phi(\mathbf{x}_n) - \mathbf{m}_i\|^2$ using (2.3)
4: Find $c^*(\mathbf{x}_n) = \arg\min_i \left(\|\phi(\mathbf{x}_n) - \mathbf{m}_i\|^2\right)$
5: **for all** clusters $\mathcal{C}_i$ $i = 1$ to k **do**
6: Update cluster $\mathcal{C}_i = \{\mathbf{x}_n | c^*(\mathbf{x}_n) = i\}$
7: **if** converged **then**
8: **return** final clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ and E calculated using (2.2)
9: **else**
10: Go to step 1

Algorithm 9: Kernel k-Means.

Table 2.1: Examples of kernel functions.

Polynomial Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j + \gamma\right)^{\delta}$
Gaussian Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2\right)$
Sigmoid Kernel	$K(\mathbf{x}_i, \mathbf{x}_j) = \tanh\left(\gamma(\mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j) + \theta\right)$

space that k-Means suffers from.

The objective function that Kernel k-Means tries to minimize is the equivalent to the clustering error in feature space shown in (2.2). We can define a *kernel matrix* $K \in \Re^{N \times N}$ where $K_{ij} = \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x}_j)$ and by taking advantage of the *kernel trick* we can compute the squared Euclidean distances in (2.2) without explicit knowledge of the transformation ϕ using (2.3). Any positive semidefinite matrix can be used as a kernel matrix since it can be interpreted as a Gram matrix. Notice that in this case cluster centers $\mathbf{m}_k$ in feature space cannot be calculated. Usually a *kernel function* $K(\mathbf{x}_i, \mathbf{x}_j)$ is used to directly provide the inner products in feature space without explicitly defining transformation ϕ (for certain kernel functions the corresponding transformation is intractable). Some kernel function examples are given in Table 2.1; $K(\mathbf{x}_i, \mathbf{x}_j) = K_{ij}$. The steps of Kernel k-Means are shown in Algorithm 9.

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{n=1}^{N} \sum_{k=1}^{M} I(\mathbf{x}_n \in \mathcal{C}_k) \|\phi(\mathbf{x}_n) - \mathbf{m}_k\|^2, \text{ where } \mathbf{m}_k = \frac{\sum_{n=1}^{N} I(\mathbf{x}_n \in \mathcal{C}_k) \phi(\mathbf{x}_n)}{\sum_{n=1}^{N} I(\mathbf{x}_n \in \mathcal{C}_k)} \tag{2.2}$$

$$\|\phi(\mathbf{x}_n) - \mathbf{m}_k\|^2 = K_{nn} - \frac{2 \sum_{j=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) K_{nj}}{\sum_{j=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k)} + \frac{\sum_{j=1}^{N} \sum_{l=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) I(\mathbf{x}_l \in \mathcal{C}_k) K_{jl}}{\sum_{j=1}^{N} \sum_{l=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) I(\mathbf{x}_l \in \mathcal{C}_k)} \tag{2.3}$$

It can be shown that Kernel k-Means monotonically converges if the kernel matrix is positive semidefinite i.e. is a *valid* kernel matrix whose entries can be interpreted as

the pairwise inner products of the dataset points in feature space. If the kernel matrix is not positive semidefinite i.e. is not a valid kernel matrix and its entries do not represent inner products, the algorithm may still converge but this is not guaranteed. As for the computational complexity, in [8] it is shown that Kernel k-Means requires $O(N^2\tau)$ scalar operations, where τ is the number of iterations until convergence is achieved. If we also have to calculate the kernel matrix an extra $O(N^2d)$ scalar operations are required.

It must be noted that, by associating a weight with each data point, the weighted Kernel k-Means algorithm is derived [7]. It has been proved that its objective function is equivalent to that of many graph partitioning problems such as ratio association, normalized cut etc if the weights and kernel are set appropriately [7, 8, 9]. This subject is further analyzed in chapter 4.

It has also been proved that performing k-Means in the kernel pca space is equivalent to Kernel k-Means [17]. However, this approach has two drawbacks: it requires computation of the eigenvectors of the kernel matrix and it highly depends on the initialization of k-Means. Calculating the eigenvectors of large matrices is expensive and even prohibitive as it requires $O(N^3)$ operations. Finally, a soft version of the Kernel k-Means algorithm has been proposed [15].

Chapter 3

The Global Kernel k-Means Algorithm

In this chapter the proposed, kernel-based, *Global Kernel k-Means* algorithm is presented and its complexity is analyzed. Moreover, two variants that accelerate its execution are introduced.

3.1 Global Kernel k-Means

The *Global Kernel k-Means* algorithm minimizes the clustering error in feature space, defined in (2.2). Our method builds on the ideas of the Global k-Means and Kernel k-Means algorithms. Global Kernel k-Means maps the dataset points from input space to a higher dimensional feature space with the help of a kernel matrix $K \in \Re^{N \times N}$ as Kernel k-Means does. In this way *nonlinearly separable clusters* are found in input space. Also, Global Kernel k-Means finds *near optimal solutions* to the M-clustering problem by incrementally and deterministically adding a new cluster center at each stage and by applying Kernel k-Means as a local search procedure instead of initializing all M clusters at the beginning of the execution. Thus the problems of *initializing the cluster centers* and getting trapped in *poor local minima* are also avoided. In a nutshell, Global Kernel k-Means combines the advantages of both Global k-Means (near optimal solutions) and Kernel k-Means (clustering in feature space).

Suppose we want to solve the M-clustering problem using Global Kernel k-Means. Since the calculation of the cluster centers in feature space is intractable, for the same reason as for Kernel k-Means, we will represent a cluster in terms of the data points that belong to it instead of its center. We start by solving the 1-clustering problem using

Input: Kernel matrix K, Number of clusters M
Output: Final clustering of the points $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$

// There is no need to solve for 1 cluster as the solution is trivial and optimal. $\mathcal{C}_1^* = \mathcal{X}$
1: **for all** k-clustering problems $k = 2$ to M **do**
2: **for all** points $\mathbf{x}_n$ $n = 1, \ldots, N$ **do** // Suppose $\mathbf{x}_n \in \mathcal{C}_r^*$
3: Run Kernel k-Means with: **input** $\left(K, k, \mathcal{C}_1^*, \ldots, \mathcal{C}_r = \mathcal{C}_r^* - \{\mathbf{x}_n\}, \ldots, \mathcal{C}_{k-1}^*, \ \mathcal{C}_k = \{\mathbf{x}_n\}\right)$ **output** $(\mathcal{C}_1^n, \ldots, \mathcal{C}_k^n, E_k^n)$
4: Find $E_k^* = \min_n(E_k^n)$ and set $(\mathcal{C}_1^*, \ldots, \mathcal{C}_k^*)$ to the partitioning corresponding to E_k^*
// This is the solution with k clusters
5: **return** $\mathcal{C}_1 = \mathcal{C}_1^*, \ldots, \mathcal{C}_M = \mathcal{C}_M^*$ as output of the algorithm

Algorithm 10: Global Kernel k-Means.

Kernel k-Means. The optimal solution to this problem is trivial as all data points are assigned in the same cluster. We continue with the 2-clustering problem where Kernel k-Means is executed N times. During the n-th execution the initialization is done by considering two clusters one of which contains only $\mathbf{x}_n$ and the other is $\mathcal{X} - \{\mathbf{x}_n\}$. Among the N solutions the one with the lowest clustering error is kept as the solution with 2 clusters. In general, for the k-clustering problem let $\left(\mathcal{C}_1^*, \ldots, \mathcal{C}_{k-1}^*\right)$ denote the solution with $k-1$ clusters and assume that $\mathbf{x}_n \in \mathcal{C}_r^*$. We perform N executions of the Kernel k-Means algorithm, with $\left(\mathcal{C}_1^*, \ldots, \mathcal{C}_r = \mathcal{C}_r^* - \{\mathbf{x}_n\}, \ldots, \mathcal{C}_{k-1}^*, \mathcal{C}_k = \{\mathbf{x}_n\}\right)$ as initial clusters for the n-th run, and keep the one resulting in the lowest clustering error. The above procedure is repeated until $k = M$. The steps of Global Kernel k-Means are shown in Algorithm 10.

The rationale behind the proposed method is based on the assumption that a near optimal solution with k clusters can be obtained through local search starting from a state with $k-1$ near optimally defined clusters (solution of the $(k-1)$-clustering problem) and the k-th cluster initialized appropriately. As we consider only one data point belonging to the k-th cluster when it is initialized, this is equivalent to initializing, during the n-th run, the k-th cluster center at point $\phi(\mathbf{x}_n)$ in feature space. Limiting the set of possible positions for the k-th center only to dataset points when mapped to feature space seems reasonable. Our experiments verify that the proposed algorithm computes near optimal solutions although it is difficult to prove theoretically. Note that during the execution of the algorithm, also solutions for every k-clustering problem with $k < M$ are obtained without additional cost, which may be desirable in case we want to decide on the number of clusters for our problem.

3.1.1 Computational Comlexity

Due to its close relation to Global k-Means and Kernel k-Means, the Global Kernel k-Means algorithm inherits their high computational cost. Given the values of the kernel matrix, the demanding step of the Kernel k-Means algorithm is the calculation of the distance between each point in feature space to every center, given by (2.3), in order to

Input: Kernel matrix K, Number of clusters M
Output: Final clustering of the points $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$

// Assume that Kernel k-Means outputs the distance of each point to each cluster center d_k^i
1: Run Kernel k-Means with: **input** $(K, 1, \mathcal{C}_1 = \mathcal{X})$, **output** $\left(\mathcal{C}_1^*, d_1^1, \ldots, d_1^N, E\right)$
// This is the solution with 1 clusters
2: **for all** k-clustering problems $k = 2$ to M **do**
3: **for all** points $\mathbf{x}_n$ $n = 1, \ldots, N$ **do**
4: Calculate $b_k^n = \sum_{i=1}^{N} \max\left(d_{k-1}^i - (K_{nn} + K_{ii} - 2K_{ni}), 0\right)$
5: Find the point with maximum b_k^n, $n^* = \arg\max_n b_k^n$ // Suppose $\mathbf{x}_{n^*} \in \mathcal{C}_r^*$
6: Run Kernel k-Means with: **input** $\left(K, k, \mathcal{C}_1^*, \ldots, \mathcal{C}_r = \mathcal{C}_r^* - \{\mathbf{x}_{n^*}\}, \ldots, \mathcal{C}_{k-1}^*, \ \mathcal{C}_k = \{\mathbf{x}_{n^*}\}\right)$
output $\left(\mathcal{C}_1^*, \ldots, \mathcal{C}_k^*, d_k^1, \ldots, d_k^N, E_k^*\right)$
// This is the solution with k clusters
7: **return** $\mathcal{C}_1 = \mathcal{C}_1^*, \ldots, \mathcal{C}_M = \mathcal{C}_M^*$ as output of the algorithm

Algorithm 11: Fast Global Kernel k-Means.

find the closest center. This is repeated for a number of iterations τ until convergence is achieved. As shown in [8] the complexity of Kernel k-Means is $O(N^2\tau)$ scalar operations. In the Global Kernel k-Means algorithm, in order to solve the M-clustering problem we must run Kernel k-Means MN times. This makes the complexity of Global Kernel k-Means $O(N^3M\tau)$. If we also have to calculate the kernel matrix an extra $O(N^2d)$ scalar operations are required making the overall complexity $O(N^2(NM\tau + d))$. Storage of the matrix requires $O(N^2)$ memory and a scheme for dealing with insufficient memory has been proposed in [29] which can be readily applied to our algorithm. As this complexity is high for large datasets two speeding up schemes are considered next.

3.2 Speeding-up Execution

Based on the general idea of the Global Kernel k-Means algorithm, several heuristics can be devised to reduce the computational load without significantly affecting the quality of the solution. In the following subsections two modifications are proposed.

3.2.1 Fast Global Kernel k-Means

The Fast Global Kernel k-Means algorithm is a simple method for lowering the complexity of the original algorithm. It is based on the same ideas as the Fast Global k-Means variant proposed in [18]. We significantly reduce the complexity by overcoming the need to execute Kernel k-Means N times when solving the k-clustering problem given the solution for the $(k-1)$-clustering problem.

Specifically, Kernel k-Means is employed *only once* and the k-th cluster is initialized to include the point $\mathbf{x}_n$ that *guarantees* the greatest reduction in clustering error. In more detail, we compute an *upper bound* $E_k^n \leq E_{k-1}^* - b_k^n$ of the final clustering error when the

k-th cluster is initialized to include point $\mathbf{x}_n$. E^*_{k-1} is the clustering error corresponding to the $(k-1)$-clustering problem solution and b^n_k measures the guaranteed reduction of the error and is defined in (3.1), where d^i_{k-1} denotes the squared distance between $\mathbf{x}_i$ and its cluster center in feature space after solving the $(k-1)$-clustering problem. We select as initialization for the k-th cluster the point $\mathbf{x}_n$ with the highest b^n_k value:

$$b^n_k = \sum_{i=1}^{N} \max\left(d^i_{k-1} - \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2, 0\right), \text{ where } \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2 = K_{nn} + K_{ii} - 2K_{ni} \tag{3.1}$$

The correctness of the above upper bound is derived from the following two facts. First, when the k-th cluster is initialized to include point $\mathbf{x}_n$ it will allocate all points that are closer to $\mathbf{x}_n$ in feature space than to their cluster center in the solution with $k-1$ clusters (distance d^i_{k-1}). Therefore, for each such point $\mathbf{x}_i$, the clustering error will decrease by $d^i_{k-1} - \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2$. Quantity b^n_k measures the reduction in error due to this reallocation. Second, since Kernel k-Means monotonically converges as long as the kernel matrix is positive semidefinite (i.e. valid), we are sure that the error will never exceed our bound. Note that the point with maximum b^n_k does not guarantee that the corresponding final clustering error E^n_k is the lowest among all final clustering errors $E^1_k, \ldots, E^N_k$. It simply guarantees the lowest upper bound. The pseudo code of the method is shown in Algorithm 11.

Computational Complexity

When using this variant of the Global Kernel k-Means algorithm in order to solve the M-clustering problem we must execute Kernel k-Means M times instead of MN times. Given the kernel matrix, calculation of b^n_k requires $O(N)$ scalar operations as d^i_{k-1} is calculated when executing Kernel k-Means for the $(k-1)$-clustering problem. Each time we have to calculate the N quantities b^n_k and this must be repeated M times in order to solve the problem with M clusters. Thus the overall cost incurred by the need to estimate the upper bound is $O(N^2M)$. Overall the Fast Global Kernel k-Means algorithm has $O(N^2(M\tau + d + M)) = O(N^2(M\tau + d))$ complexity, which is considerably lower than that of Global Kernel k-Means and is comparable to the complexity of Kernel k-Means when M is sufficiently small. In general, this reduction in complexity comes at the cost of finding solutions with higher clustering error than the original algorithm, due to the greedy decision made when deciding on the best point for initializing the newly added cluster. As our experiments indicate, there are several cases where the performance of the fast version is similar to that of Global Kernel k-Means which makes it a good fast alternative.

3.2.2 Global Kernel k-Means with Convex Mixture Models

Another way, apart from Fast Global Kernel k-Means, to lower the complexity of the Global Kernel k-Means algorithm when solving the M-clustering problem is to select a

set of good exemplars in feature space with an exemplar-based method and then apply Global Kernel k-Means, with the restriction that the points tried as initializations for the newly added cluster come from this set only.

In order to solve the k-clustering problem given the solution with $k-1$ clusters we must run Kernel k-Means as many times as the number of selected exemplars instead of N times. In each of these runs we initialize the k-th cluster center to an exemplar of those selected. Many algorithms that perform exemplar-based clustering can be used for this purpose such as affinity propagation [11]. In our work we integrate the Global Kernel k-Means algorithm with the one proposed by Lashkari et al. [16], which is based on the use of exemplar-based mixture models for clustering. We will refer to this algorithm as *convex mixture model (CMM) clustering.*

As already discussed in section 1.2.4, clustering with mixture models results in soft assignments of data points to clusters in such a way that the likelihood of the mixture model is maximized. Given a dataset $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, $\mathbf{x}_i \in \Re^d$ we seek to maximize the following log-likelihood function:

$$L\left(\{q_j\}_{j=1}^M, \{\boldsymbol{\mu}_j\}_{j=1}^M; \mathcal{X}\right) = \frac{1}{N}\sum_{i=1}^{N} \log\left[\sum_{j=1}^{M} q_j f(\mathbf{x}_i; \boldsymbol{\mu}_j)\right] \tag{3.2}$$

where q denotes the prior probability of a component satisfying the constraint $\sum_{j=1}^M q_j = 1$ and $f(\mathbf{x}; \boldsymbol{\mu})$ is an exponential family distribution on random variable $\mathbf{X}$. Note that the above log-likelihood implies a mixture model whose components are exponential family distributions. We restrict ourselves to this special case of mixture models as the Lashkari et al. [16] algorithm was derived based on them. Taking into account the bijection between regular exponential families and Bregman divergences [3] we can rewrite the above exponential family distribution as $f(\mathbf{x}; \boldsymbol{\mu}) = C(\mathbf{x}) \exp(-d_\varphi(\mathbf{x}, \boldsymbol{\mu}))$, where d_φ is some Bregman divergence and $C(\mathbf{x})$ is independent of $\boldsymbol{\mu}$. Note that with this representation $\boldsymbol{\mu}$ is the expected value of random variable $\mathbf{X}$ under the distribution $f(\mathbf{x}; \boldsymbol{\mu})$. If we take Euclidean distance as the Bregman divergence we obtain a Gaussian mixture model.

Usually the maximization of the above likelihood function is done with the EM algorithm. This algorithm locates local maxima of the objective function and is sensitive to initialization. This sensitivity causes a lot of trouble in finding a good solution especially when the mixture model has many components i.e. many q and $\boldsymbol{\mu}$ quantities to initialize. The most common approach to avoid poor local maxima is to run the algorithm many times with different random initializations and keep the best solution. Lashkari et al. [16] try to circumvent the initialization procedure by introducing an exemplar-based likelihood function that approximates the exact likelihood and results in a *convex* minimization problem whose globally optimal solution can be found with a simple algorithm.

In order to achieve this, they take models of the form (3.2) and restrict the set of mixture components to the distributions centered at the data points i.e. $\boldsymbol{\mu}_j \in \mathcal{X}$. Furthermore, they increase the number of the mixture components to N and thus represent all data points as cluster center candidates (candidate exemplars). The proposed log-

likelihood function is:

$$L\left(\{q_j\}_{j=1}^N;\mathcal{X}\right)=\frac{1}{N}\sum_{i=1}^N\log\left[\sum_{j=1}^N q_j f_j(\mathbf{x}_i)\right]=\frac{1}{N}\sum_{i=1}^N\log\left[\sum_{j=1}^N q_j e^{-\beta d_\varphi(\mathbf{x}_i,\mathbf{x}_j)}\right]+\text{const.} \tag{3.3}$$

where $f_j(\mathbf{x})$ is an exponential family distribution with its expectation parameter equal to the j-th data point. The constant β controlls the sharpness of the components and $0<\beta<\infty$. In case of a Gaussian mixture model β is equivalent to a common fixed precision for all the components. Moreover this parameter *controls the number of clusters* identified by the algorithm when the soft assignments are turned into hard ones. Higher β values result in more clusters in the final solution. To solve the above optimization problem we maximize $L\left(\{q_j\}_{j=1}^N;\mathcal{X}\right)$ over $\{q_j\}_{j=1}^N$ s.t. $\sum_{j=1}^N q_j=1$ which are the only unknown parameters of the simplified likelihood function.

The log-likelihood function (3.3) can be expressed in terms of the Kullback-Leibler (KL) divergence if we define $\hat{P}(\mathbf{x})=1/N,\mathbf{x}\in\mathcal{X}$ to be the empirical distribution of the dataset, $Q(\mathbf{x})=\sum_{j=1}^N q_j f_j(\mathbf{x})$ the mixture model distribution and by noting that

$$D(\hat{P}\|Q)=-\sum_{\mathbf{x}\in\mathcal{X}}\hat{P}(\mathbf{x})\log Q(\mathbf{x})-\mathbb{H}(\hat{P})=-L\left(\{q_j\}_{j=1}^N;\mathcal{X}\right)+\text{const.} \tag{3.4}$$

where $\mathbb{H}(\hat{P})$ is the entropy of the empirical distribution that does not depend on the parameters of the mixture model. Now the maximization of (3.3) is equivalent to the minimization of (3.4). This minimization problem is *convex* and can be solved with an efficient-iterative algorithm. Due to the convexity we will refer to mixture models of the form (3.3) as *convex mixture models*. As proved in [6] the updates on the components' prior probabilities are given by:

$$q_j^{(t+1)}=q_j^{(t)}\sum_{\mathbf{x}\in\mathcal{X}}\frac{\hat{P}(\mathbf{x})f_j(\mathbf{x})}{\sum_{j'=1}^N q_{j'}^{(t)}f_{j'}(\mathbf{x})} \tag{3.5}$$

and the algorithm is guaranteed to converge to the global minimum as long as $q_j^{(0)}>0\ \forall j$. The prior probability q_j associated with data point $\mathbf{x}_j$ is a measure of *how likely this point is to be an exemplar*. This is an important fact for our integration with Global Kernel k-Means.

The convex mixture model whose likelihood is described in (3.3) performs clustering in input space. For our purpose we need to alter it a bit so as to perform clustering in feature space. For this reason we use the same nonlinear transformation ϕ as in Kernel k-Means to map the data points to feature space and the log-likelihood (3.3) becomes:

$$L\left(\{q_j\}_{j=1}^N;\mathcal{X}\right)=\frac{1}{N}\sum_{i=1}^N\log\left[\sum_{j=1}^N q_j f_j(\phi(\mathbf{x}_i))\right]=\frac{1}{N}\sum_{i=1}^N\log\left[\sum_{j=1}^N q_j e^{-\beta d_\varphi(\phi(\mathbf{x}_i),\phi(\mathbf{x}_j))}\right]+\text{const.} \tag{3.6}$$

where $f_j(\phi(\mathbf{x}))$ is an exponential family distribution with its expectation parameter equal to the mapping of the j-th data point in feature space. If we select Euclidean distance

as the Bregman divergence, as in our experiments, then we obtain a Gaussian convex mixture model in feature space and $d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$ can be expressed in terms of a kernel function as:

$$d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) = \|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2 = K_{ii} + K_{jj} - 2K_{ij} \tag{3.7}$$

Given the convex mixture model corresponding to the likelihood defined in (3.6) we now proceed to its integration with Global Kernel k-Means, leading to the Global Kernel k-Means with CMM algorithm. Let as note that to get the exemplars in feature space the likelihood is optimized only once. For the implementation of the algorithm that optimizes (3.6) we follow the same steps with Laskari et al. [16]. Letting $s_{ij} = \exp(-\beta d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)))$ and using two auxiliary vectors z, n we update the prior probabilities q_j as follows:

$$z_i^{(t)} = \sum_{j=1}^{N} s_{ij} q_j^{(t)}, \quad n_j^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{s_{ij}}{z_i^{(t)}}, \quad q_j^{(t+1)} = n_j^{(t)} q_j^{(t)} \tag{3.8}$$

where $q_j^{(0)} > 0$ for all data points we want to consider as possible exemplars. As our target is to identify a number of good exemplars, say P exemplars, we run the algorithm until the P highest q_j values correspond to the same data points in feature space for a number of consecutive iterations. Moreover we require that the order among the P highest q_j values remains the same during these iterations. Note that this convergence criterion differs from that proposed in [16]. The points of the dataset that correspond to the P highest q_j values are those considered as good exemplars.

After we have determined the set of exemplars we run Global Kernel k-Means. The set $\mathcal{X}'$ contains the dataset points whose representation in feature space is among the P exemplars. Thus $\mathcal{X}' \subset \mathcal{X}$ and $\mathcal{X}' = \{\mathbf{x}_j | q_j > q_{thres}\}$ where q_{thres} is the $P+1$ highest prior value. We will refer to the data points belonging to $\mathcal{X}'$ as $\mathbf{x}'_j$ $j = 1, \ldots, P$. Now suppose we want to solve the M-clustering problem. We must solve all intermediate problems with $1, \ldots, M$ clusters but now instead of trying N different initializations for the newly added cluster, one for each point in $\mathcal{X}$, for each intermediate problem we try P initializations, one for each point in $\mathcal{X}'$. In more detail, for the k-clustering problem let $\left(\mathcal{C}_1^*, \ldots, \mathcal{C}_{k-1}^*\right)$ denote the solution with $k-1$ clusters and assume that $\mathbf{x}'_p \in \mathcal{C}_r^*$. We perform P executions of the Kernel k-Means algorithm with $\left(\mathcal{C}_1^*, \ldots, \mathcal{C}_r = \mathcal{C}_r^* - \{\mathbf{x}'_p\}, \ldots, \mathcal{C}_{k-1}^*, \mathcal{C}_k = \{\mathbf{x}'_p\}\right)$ as initial clusters for the p-th run, and keep the one resulting in the lowest clustering error as the solution with k clusters. Remember that initializing cluster $\mathcal{C}_k$ to contain point $\mathbf{x}'_p$ during the p-th run is the same as placing the k-th center at point $\phi(\mathbf{x}'_p)$ in feature space. The above procedure is repeated for $k = 2, \ldots, M$. The steps of the Global Kernel k-Means with CMM algorithm are shown in Algorithm 12 where the use of a Gaussian convex mixture model is assumed. It is obvious that this variant is also a kernel-based clustering method. If the distances $d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$ are given by a distance matrix or they can be calculated through the kernel matrix, as for squared Euclidean distances, then the dataset in vectorial form is not required.

Input: Kernel matrix K, Number of clusters M, Number of exemplars P, Parameter β
Output: Final clustering of the points $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_M$

// Convex mixture model algorithm
1: **for all** points $\mathbf{x}_i$ $i = 1, \ldots, N$ **do**
2: **for all** points $\mathbf{x}_j$ $j = 1, \ldots, N$ **do**
3: Calculate $s_{ij} = \exp(-\beta(K_{ii} + K_{jj} - 2K_{ij}))$
4: $t = 0$
5: **for all** components $j = 1, \ldots, N$ **do**
6: Initialize prior probabilities $q_j^{(0)} = \frac{1}{N}$
7: **for** $i = 1, \ldots, N$ **do**
8: Update auxiliary vector z, $z_i^{(t)} = \sum_{j=1}^{N} s_{ij} q_j^{(t)}$
9: **for** $j = 1, \ldots, N$ **do**
10: Update auxiliary vector n, $n_j^{(t)} = \frac{1}{N} \sum_{i=1}^{N} \frac{s_{ij}}{z_i^{(t)}}$
11: **for all** components $j = 1, \ldots, N$ **do**
12: Update prior probabilities $q_j^{(t+1)} = n_j^{(t)} q_j^{(t)}$
13: **if** converged **then**
14: Calculate $\mathcal{X}' = \{\mathbf{x}_j | q_j > q_{thres}\}$ where q_{thres} is the $P+1$ highest prior value, Go to step 17
15: **else**
16: $t = t + 1$, Go to step 7
// Global Kernel k-Means algorithm
// There is no need to solve for 1 cluster as the solution is trivial and optimal. $\mathcal{C}_1^* = \mathcal{X}$
17: **for all** k-clustering problems $k = 2$ to M **do**
18: **for all** points $\mathbf{x}'_p$ $p = 1, \ldots, P$ **do** // Suppose $\mathbf{x}'_p \in \mathcal{C}_r^*$
19: Run Kernel k-Means with: **input** $\left(K, k, \mathcal{C}_1^*, \ldots, \mathcal{C}_r = \mathcal{C}_r^* - \{\mathbf{x}'_p\}, \ldots, \mathcal{C}_{k-1}^*, \ \mathcal{C}_k = \{\mathbf{x}'_p\}\right)$ **output** $\left(\mathcal{C}_1^p, \ldots, \mathcal{C}_k^p, E_k^p\right)$
20: Find $E_k^* = \min_p(E_k^p)$ and set $(\mathcal{C}_1^*, \ldots, \mathcal{C}_k^*)$ to the partitioning corresponding to E_k^*
// This is the solution with k clusters
21: **return** $\mathcal{C}_1 = \mathcal{C}_1^*, \ldots, \mathcal{C}_M = \mathcal{C}_M^*$ as output of the algorithm

Algorithm 12: Global Kernel k-Means with CMM.

Clustering with the convex mixture model corresponding to likelihood (3.6) requires to select a value for the β parameter. As already mentioned this parameter controls the sharpness of the components and $0 < \beta < \infty$. Neither extreme, one selecting a small value and thus assigning all points to the central exemplar and the other selecting a large value and thus taking all points as exemplars, when the soft assignments are turned into hard ones, is interesting. It is possible to identify a reasonable range of β values by determining a reference value β_0. For this purpose we choose the empirical value proposed in [16] but appropriately modified for clustering in feature space: $\beta_0 = N^2 \log N / \sum_{i,j} d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$. Usually a β value around β_0 will give good clustering results. In most of our experiments we set $\beta = \beta_0$ and obtain good solutions.

The convergence of the algorithm described in (3.8) can be improved if we identify after each iteration the prior probabilities q_j that are below a small threshold, set them to

zero and renormalize over the remaining indices. This way we exclude the corresponding data points from being selected as exemplars. In our experiments we set this threshold to $10^{-3}/N$.

When using this variant of Global Kernel k-Means we select P exemplars in order to solve the M-clustering problem. *The idea behind this is that the exemplars identified by the convex mixture model contain most of the information required by Global Kernel k-Means to partition the dataset.* So by trying only the P exemplars as possible initializations for the newly added cluster we hope to approximate the solution of the original Global Kernel k-Means algorithm. Usually one should choose $P > M$ but $P << N$ because in most cases only a small subset of the dataset contains useful information for its partitioning.

Computational Complexity

With this variant, we must run Kernel k-Means MP times instead of MN times to solve the M-clustering problem. This is a great reduction, especially for large datasets, if we consider that $P << N$ for the aforementioned reasons. Of course there is the additional cost of identifying the exemplars. The algorithm (3.8) considered here has a complexity of $O(N^2)$ per iteration [16]. Moreover the calculation of the quantities s_{ij} requires an additional $O(N^2)$ operations if the kernel matrix is given. If the algorithm requires τ' iterations to converge the overall cost becomes $O(N^2\tau')$. The Global Kernel k-Means algorithm has a cost of $O(N^2(PM\tau + d))$ in this case including the kernel calculation. So the overall cost of the proposed variant is $O(N^2(PM\tau + d + \tau'))$. This complexity is considerably lower to that of the original algorithm when $P << N$. Our experiments indicate that despite this reduction to the complexity, the quality of the solutions is satisfactory and very similar to that of the original algorithm in many cases. Finally, the complexity of Fast Global Kernel k-Means is lower but this algorithm finds better solutions that are closer to those of the original algorithm as shown in our experiments. This happens because this method is less greedy as it tries P initializations when solving the k-clustering problem instead of one.

Chapter 4

Weighted Global Kernel k-Means and Graph Cuts

This chapter presents a weighted version of the Global Kernel k-Means algorithm and its two speeding up schemes based on the weighted Kernel k-Means algorithm. The weighted versions of the algorithms allow us to prove an equivalence of their objective with many popular graph cut criteria and we discuss how Global Kernel k-Means can be used for graph clustering.

4.1 Weighted Kernel k-Means

If we associate a positive weight with each data point the weighted Kernel k-Means algorithm is derived [7]. The weights play a crucial role in proving an equivalence of clustering to graph partitioning which is the reason we present this version of Kernel k-Means. Again, suppose we want to solve the M-clustering problem. The objective function is expressed as follows, where $w_i > 0$ is the weight associated with data point $\mathbf{x}_i$:

$$E\left(\mathbf{m}_1, \ldots, \mathbf{m}_M\right) = \sum_{i=1}^{N}\sum_{k=1}^{M} I(\mathbf{x}_i \in \mathcal{C}_k) w_i \|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2, \text{ where } \mathbf{m}_k = \frac{\sum_{i=1}^{N} I(\mathbf{x}_i \in \mathcal{C}_k) w_i \phi(\mathbf{x}_i)}{\sum_{i=1}^{N} I(\mathbf{x}_i \in \mathcal{C}_k) w_i} \tag{4.1}$$

Note that the center of the cluster in feature space is the weighted average of the points that belong to the cluster. Once again we can take advantage of the kernel trick and calculate the squared Euclidean distances in (4.1) without explicitly defining transformation

ϕ using (4.2). Algorithm 9 can be applied for this version of Kernel k-Means by only modifying lines 3 and 8 where the point to cluster center distances must be calculated using (4.2) and the clustering error using (4.1).

$$\|\phi(\mathbf{x}_i) - \mathbf{m}_k\|^2 = K_{ii} - \frac{2\sum_{j=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) w_j K_{ij}}{\sum_{j=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) w_j} + \frac{\sum_{j=1}^{N}\sum_{l=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) I(\mathbf{x}_l \in \mathcal{C}_k) w_j w_l K_{jl}}{\sum_{j=1}^{N}\sum_{l=1}^{N} I(\mathbf{x}_j \in \mathcal{C}_k) I(\mathbf{x}_l \in \mathcal{C}_k) w_j w_l} \tag{4.2}$$

4.2 Expanding Weights to Global Kernel k-Means and its Variants

4.2.1 Weighted Global Kernel k-Means

It is straightforward that we can use the Global Kernel k-Means algorithm to optimize the objective function defined in (4.1) by associating a weight with each data point. Algorithm 10 can be applied with the slightest modification to get the weighted Global Kernel k-Means algorithm. Specifically, we must run weighted Kernel k-Means instead of Kernel k-Means in line 3 and on the input to include the weights. All other steps remain the same.

4.2.2 Weighted Fast Global Kernel k-Means

Fast Global Kernel k-Means can also be extended to handle weights for each data point. Again we must run weighted Kernel k-Means instead of Kernel k-Means but the issue that changes is the way we select the point that guarantees the greatest reduction in clustering error when solving the k-clustering problem given the solution with $k-1$ clusters. This point will be used to initialize the k-th cluster for the one and only run of weighted Kernel k-Means. Now the weights must be taken into account, resulting in a modified definition of the quantity b_k^n (4.3) that measures the reduction in clustering error when the k-th cluster is initialized to include point $\mathbf{x}_n$. d_{k-1}^i again denotes the squared distance between $\mathbf{x}_i$ and its cluster center in feature space after solving the $(k-1)$-clustering problem.

$$b_k^n = \sum_{i=1}^{N} w_i \max\left(d_{k-1}^i - \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2, 0\right)$$
$$\text{where} \quad \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2 = K_{nn} + K_{ii} - 2K_{ni} \tag{4.3}$$

The above definition measures the reduction in clustering error by identifying the points that will be allocated by the new cluster center. For each such point $\mathbf{x}_i$, the clustering error will decrease by $w_i(d_{k-1}^i - \|\phi(\mathbf{x}_n) - \phi(\mathbf{x}_i)\|^2)$. Quantity b_k^n measures the reduction in clustering error due to this reallocation. This reduction is guaranteed as weighted Kernel k-Means monotonically converges if the kernel matrix is positive semidefinite (valid). Apparently, we select as initialization for the k-th cluster the point $\mathbf{x}_n$ that maximizes b_k^n. No other modifications are required to obtain weighted Fast Global Kernel k-Means.

4.2.3 Weighted Global Kernel k-Means with Convex Mixture Models

The Global Kernel k-Means with CMM algorithm is divided in two parts. The first is the identification of good exemplars and the second the application of Global Kernel k-Means where for the newly added cluster we try only the exemplars as possible initializations. It is obvious that weights can be applied to the second part, as there we run Global Kernel k-Means which handles weights as shown in section 4.2.1. Now we turn our attention to the first part and propose a modification of the convex mixture model algorithm in order to accommodate weights. We can incorporate the weights into the convex mixture model objective through the empirical distribution of the dataset. Instead of a uniform distribution we define:

$$\hat{P}(\mathbf{x}) = \begin{cases} \frac{w_i}{\sum_{i'=1}^{N} w_{i'}} & \mathbf{x} \in \mathcal{X} \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

If $Q(\mathbf{x}) = \sum_{j=1}^{N} q_j f_j(\phi(\mathbf{x}))$ is the convex mixture model distribution in feature space we can write the convex mixture model objective in terms of the KL divergence as:

$$D(\hat{P}\|Q) = -\sum_{\mathbf{x}\in\mathcal{X}} \hat{P}(\mathbf{x}) \log Q(\mathbf{x}) - \mathbb{H}(\hat{P}) \tag{4.5}$$

This is the same objective as for the unweighted case but with (4.4) in place of the empirical distribution. It can be proved that the minimization of (4.5) can be performed in the same way as in the unweighted case, thus the updates on the components' priors probabilities are given by:

$$q_j^{(t+1)} = q_j^{(t)} \sum_{\mathbf{x}\in\mathcal{X}} \frac{\hat{P}(\mathbf{x}) f_j(\phi(\mathbf{x}))}{\sum_{j'=1}^{N} q_{j'}^{(t)} f_{j'}(\phi(\mathbf{x}))} \tag{4.6}$$

with $\hat{P}(\mathbf{x})$ defined in (4.4). We can use the algorithm described in (3.8) to update the priors with a slight modification on $n_j^{(t)}$, now defined as $n_j^{(t)} = \sum_{i=1}^{N} \hat{P}(\mathbf{x}_i) \frac{s_{ij}}{z_i^{(t)}}$.

The change on the empirical dataset distribution results in a change on the empirical value of the β parameter. We follow the same approach as in [16] and reformulate our problem as a rate-distortion problem. By making use of *proposition 1* in [16] and defining

$$Q'(j, \mathbf{x}) = q_j f_j(\phi(\mathbf{x})) = q_j C(\mathbf{x}) e^{-\beta d_\varphi(\phi(\mathbf{x}), \phi(\mathbf{x}_j))} \tag{4.7}$$

$$P'(j, \mathbf{x}) = \hat{P}(\mathbf{x}) P'(j|\mathbf{x}) = \begin{cases} \frac{w_i}{\sum_{i'=1}^{N} w_{i'}} r_{ij} & \mathbf{x} \in \mathcal{X} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

where $r_{ij} = P'(j|\mathbf{x}_i)$, the minimization of (4.5) becomes equivalent to minimizing the following objective:

$$D(P'\|Q') = \sum_{i,j=1}^{N} \hat{P}(\mathbf{x}_i) r_{ij} \left[\log \frac{r_{ij}}{q_j} + \beta d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) \right] + \text{const.} \tag{4.9}$$

Actually, since $P'(j,\mathbf{x}) \neq 0$ only for $\mathbf{x} \in \mathcal{X}$, the objective function is expressed only in terms of variables q_j and $P'(j|\mathbf{x}_i)$ therefore our goal is to minimize (4.9) in the space of distributions of random variable $(J,I) \in \{1,\ldots,N\} \times \{1,\ldots,N\}$ namely, in the product space of mixture model component indices $\times$ data point indices. For any set of values r_{ij}, setting $q_j = \sum_{i=1}^{N} \hat{P}(\mathbf{x}_i) r_{ij}$ minimizes (4.9). Substituting this on the above equation we obtain:

$$\begin{aligned} D(P' \| Q'(P')) &= \sum_{i,j=1}^{N} \hat{P}(\mathbf{x}_i) r_{ij} \left[\log \frac{r_{ij}}{\sum_{i'=1}^{N} \hat{P}(\mathbf{x}_{i'}) r_{i'j}} + \beta d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) \right] + \text{const.} \\ &= \mathbb{I}(J;I) + \beta \mathbb{E}_{J,I} d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j)) + \text{const.} \end{aligned} \tag{4.10}$$

where the first term is the mutual information under distribution $P'(j,\mathbf{x})$ and the second term is the expected value of the pairwise distances in feature space under the same distribution. Note that (4.10) is almost identical to equation (8) of Lashkari et al. [16]. The only difference is on the empirical distribution of the dataset.

If we interpret the above problem in the framework of rate distortion theory we can think of r_{ij}'s as a probabilistic encoding in feature space of the dataset onto itself with the corresponding average distortion $D = \mathbb{E}_{J,I} d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$ and the rate $\mathbb{I}(J;I)$. For the derivative of the rate-distortion function $R(D)$ [5] again we have $\partial R / \partial D = -\beta$. In order to identify a good reference value for β we follow the same approach as in [16] and define β_0 to be the slope of a line connecting the following two points on the rate distortion graph: the case of sending any point to itself with zero distortion and rate equal to the entropy of the dataset $R(0) = -\sum_{i=1}^{N} \hat{P}(\mathbf{x}_i) \log \hat{P}(\mathbf{x}_i)$ and the case of a random code, $r_{ij} = 1/N$, with zero rate and average distortion $D = \frac{1}{N} \sum_{i,j=1}^{N} \hat{P}(\mathbf{x}_i) d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))$. The empirical value of the β parameter for the weighted convex mixture model becomes:

$$\beta_0 = N \frac{-\sum_{i=1}^{N} \hat{P}(\mathbf{x}_i) \log \hat{P}(\mathbf{x}_i)}{\sum_{i,j=1}^{N} \hat{P}(\mathbf{x}_i) d_\varphi(\phi(\mathbf{x}_i), \phi(\mathbf{x}_j))} \tag{4.11}$$

In summary, to run weighted Global Kernel k-Means with CMM one must define an empirical distribution of the form (4.4), in order to incorporate the weights into the convex mixture model objective, and find the mixture components prior probabilities using the updates described in (4.6). A good range of values for the β parameter is located through β_0 defined in (4.11). After identifying the exemplars we run weighted Global Kernel k-Means as described in section 4.2.1.

4.3 Graph Partitioning

Graph partitioning is another approach to clustering data. In this problem we are given a graph $G = (\mathcal{V}, \mathcal{E}, A)$, where $\mathcal{V}$ denotes the set of vertices, $\mathcal{E}$ the set of edges and A is an $|\mathcal{V}| \times |\mathcal{V}|$ affinity matrix which contains the pairwise similarities of the vertices i.e. the weights of the edges, and we aim to partition the graph into M disjoint clusters such that $\mathcal{V}_1 \cup \mathcal{V}_2 \cup \ldots \cup \mathcal{V}_M = \mathcal{V}$.

A number of different graph partitioning objectives have been proposed such as ratio association, ratio cut, normalized cut etc. Usually spectral methods, like the one in [20], are employed to solve these problems by calculating the eigenvectors of the affinity matrix. Spectral methods perform well because they compute globally optimal solutions of a *relaxation* of the problem considered. Calculating the eigenvectors of large matrices i.e. for graphs with many vertices, is computationally expensive, as it requires $O(N^3)$ operations, and may also be infeasible. In [7, 8, 9] it is proved that the weighted Kernel k-Means objective is equivalent to that of many graph cut problems if the weights and kernel are set appropriately. The proof is based on formulating the problems as trace maximizations following a similar approach to [28], where the k-Means objective is formulated as a trace maximization. Weighted Kernel k-Means avoids the need to calculate eigenvectors but cannot find the optimal solution because it depends on cluster initialization. Even when calculation of eigenvectors is possible, the experiments in [8] show that weighted Kernel k-Means can further improve the clustering result produced by spectral methods.

As discussed in section 4.2 Global Kernel k-Means and its two variants can also be used to optimize (4.1). Hence, these algorithms can also be applied to graph partitioning and incorporate the advantages they bring over Kernel k-Means to this task. As shown in our experiments, these algorithms outperform Kernel k-Means on graph partitioning on most cases and thus can be considered as a good alternative to spectral methods, especially the two variants which combine good quality with low computational cost. Here we focus on the ratio association and normalized cut problems, which we experimented with, and present how the weights and kernel must be set to obtain the equivalence. The reader is referred to [8, 9] for a thorough analysis on the subject.

Let us denote $\text{links}(\mathcal{A}, \mathcal{B})$ to be the sum of the edge weights between nodes in $\mathcal{A}$ and $\mathcal{B}$ i.e. $\text{links}(\mathcal{A}, \mathcal{B}) = \sum_{i \in \mathcal{A}} \sum_{j \in \mathcal{B}} A_{ij}$ and $\text{degree}(\mathcal{A})$ to be the sum of the edge weights between nodes in $\mathcal{A}$ and all vertices i.e. $\text{degree}(\mathcal{A}) = \text{links}(\mathcal{A}, \mathcal{V})$. Also let D be the diagonal $|\mathcal{V}| \times |\mathcal{V}|$ degree matrix where $D_{ii} = \sum_{j=1}^{|\mathcal{V}|} A_{ij}$.

1) Ratio Association: The ratio association problem tries to maximize within cluster association relative to the cluster size. The objective function is:

$$RA(G) = \max_{\mathcal{V}_1, \ldots, \mathcal{V}_M} \sum_{i=1}^{M} \frac{\text{links}\,(\mathcal{V}_i, \mathcal{V}_i)}{|\mathcal{V}_i|} \tag{4.12}$$

As proved in [8], to make the objective function of weighted Kernel k-Means and weighted Global Kernel k-Means equivalent to that of ratio association we must set $w_i = 1$ and $K = A$. As $w_i = 1$ this is practically the unweighted version of the algorithms where the affinity matrix is in place of the kernel matrix. Obviously dataset $\mathcal{X}$ contains the nodes and $N = |\mathcal{V}|$.

2) Normalized Cut: The normalized cut problem aims to minimize the cut between clusters and the remaining vertices relative to the degree of the cluster. This objective is

one of the most popular in graph partitioning [23, 27] and is defined as:

$$NC(G) = \min_{\mathcal{V}_1,\ldots,\mathcal{V}_M} \sum_{i=1}^{M} \frac{\text{links}\,(\mathcal{V}_i, \mathcal{V}\backslash\mathcal{V}_i)}{\text{degree}(\mathcal{V}_i)} \tag{4.13}$$

As shown in [8], to make the objective function of weighted Kernel k-Means and weighted Global Kernel k-Means equivalent to that of normalized cut we must set $w_i = D_{ii}$ and $K = D^{-1}AD^{-1}$. Obviously dataset $\mathcal{X}$ contains the nodes and $N = |\mathcal{V}|$.

The above definitions of the kernel matrix do not guarantee that it will be positive semidefinite (i.e. valid), since A can be an arbitrary adjacency matrix, hence the algorithms may not converge. Remember that, positive semidefiniteness is a sufficient but not necessary condition to guarantee convergence of the discussed algorithms. A workaround for this problem is proposed in [8] which relies on a diagonal shift of the kernel matrix. For the ratio association problem we define K as: $K = \lambda I + A$ where I is the identity matrix and λ is a constant large enough to ensure that K is positive semidefinite. For the normalized cut problem we set $K = \lambda D^{-1} + D^{-1}AD^{-1}$. A good thing is that this manipulation of the kernel matrix does not change the optimization problem we solve although it can degrade the performance of the algorithms as shown in [8] when the shift is too large.

Chapter 5

Experimental Evaluation

In this section we thoroughly study the performance of Global Kernel k-Means, its two variants and Kernel k-Means with multiple restarts by applying these algorithms on many clustering problems. These problems range from artificial datasets to MRI segmentation, digits and face images clustering and graph partitioning. Our aim is to test the robustness of the proposed Global Kernel k-Means algorithm and its two variants on different tasks, discover if indeed the original algorithm avoids getting trapped in poor local minima and also how close the solutions of the speeding up schemes are to those of the original algorithm. The experiments were conducted on a computer running Windows XP with 3.5GB RAM and 2.4GHz Intel Core 2 Duo processor. The code was implemented in MATLAB[1].

5.1 Artificial Datasets

We compare the four clustering algorithms mentioned above on two artificial datasets. The setup of the experiments is as follows: we use a Gaussian kernel which has a parameter σ whose value needs to be determined a priori. When comparing the algorithms the same σ is used for all four methods so as to obtain a meaningful comparison. For the Global

[1]For better understanding the experimental results it is suggested to view the figures in color.

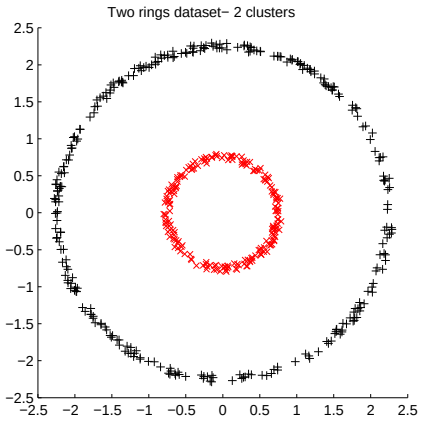


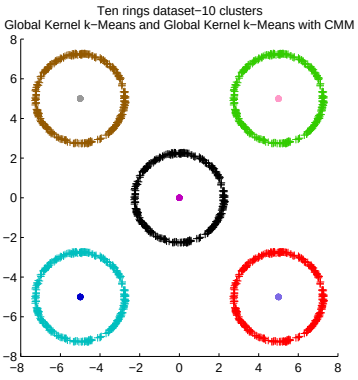
Figure 5.1: Global Kernel k-Means, Global Kernel k-Means with CMM, Fast Global Kernel k-Means and Kernel k-Means (run with minimum clustering error) on the two rings dataset.

Kernel k-Means with CMM algorithm, we set $\beta = \beta_0$, $s_{ij} = \exp(-\beta\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2)$ and select twice as many exemplars as the number of clusters ($P = 2M$). The quality of the solutions produced by the algorithms is evaluated in terms of clustering error. As Global Kernel k-Means and its two variants perform clustering deterministically they are run once. On the other hand Kernel k-Means is restarted 100 times and we report the average clustering error, its standard deviation as well as the minimum and maximum values during the 100 runs.

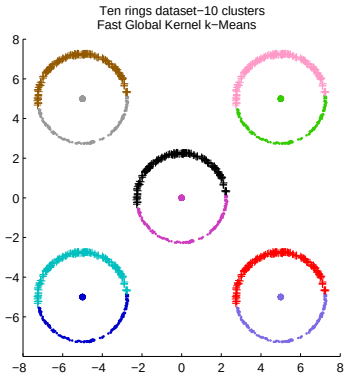
Table 5.1: Artificial datasets results in terms of clustering error. The kernel parameter σ value is also shown.

Method/Dataset		**Two Rings** $\sigma = 1$	**Ten Rings** $\sigma = 1.8$
Global Kernel k-Means		**320.17**	**966.87**
Global Kernel k-Means with CMM		**320.17**	**966.87**
Fast Global Kernel k-Means		**320.17**	1073.18
Kernel k-Means (100 runs)	**Average**	334.4	1107.97
	Std	6.4	177.24
	Min	**320.17**	981.53
	Max	351.05	1765.29

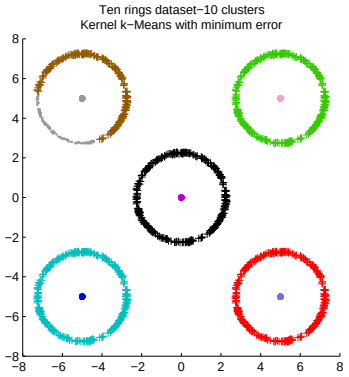
The first dataset (Figure 5.1) contains two rings with a total of 500 points and we perform clustering into 2 clusters. Obviously this problem is not linearly separable, hence any algorithm, such as k-Means, which identifies linearly separable clusters in input space is inappropriate for this task. Global Kernel k-Means and its two variants manage to



(a) Global Kernel k-Means and Global Kernel k-Means with CMM.



(b) Fast Global Kernel k-Means.



(c) Kernel k-Means (the run with minimum clustering error).

Figure 5.2: Clustering of the ten rings dataset.

identify the two rings. Kernel k-Means finds the two rings solution only in 12 out of 100 runs. The solution that appropriately splits the two rings corresponds to the lowest clustering error as can be seen in Table 5.1. This result is depicted in Figure 5.1.

The second dataset (Figure 5.2) consists of five copies of two rings where the inner ring is dense and has 700 points while the outer ring has 300 points. The whole dataset has 5000 points and 10 clusters. For this difficult clustering problem, the Global Kernel k-Means algorithm manages to discriminate ten rings as shown in Figure 5.2(a). Also Global Kernel k-Means with CMM identifies the ten rings correctly. This demonstrates that the exemplars determined through the CMM algorithm capture most of the information required to partition the dataset. The solution corresponding to those two algorithms is the one with the lowest clustering error in Table 5.1. Fast Global Kernel k-Means fails to correctly identify all the rings: it splits the outer ring into two parts and merges one of them with the inner ring as shown in Figure 5.2(b). As the problem is quite hard to solve, the greedy decisions made by this algorithm result in suboptimal solutions with higher clustering error. Finally, Kernel k-Means never splits correctly the ten rings during the 100 runs and note also that its average clustering error is higher to that of Fast Global

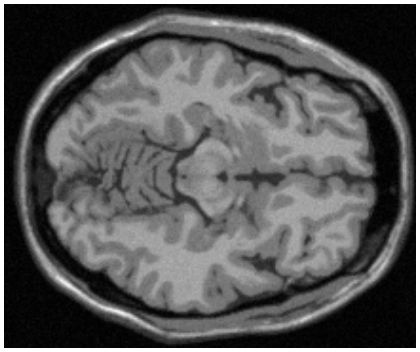
Kernel k-Means. Figure 5.2(c) depicts the clustering result corresponding to the lowest clustering error during the 100 runs of Kernel k-Means.

Overall, Global Kernel k-Means achieves the lowest clustering error for both datasets and none of the 100 runs of Kernel k-Means provides a better result. This observation together with the fact that it correctly locates the rings in the second dataset, which is a difficult problem to solve, justify our claim that it finds near optimal solutions. Also the Global Kernel k-Means with the CMM algorithm, turns out to provide identical results to the original algorithm on these artificial datasets thus making him a good alternative with lower computational cost. To further demonstrate the potential of this method we reran the experiments by selecting this time as many exemplars as the number of clusters ($P = M$) and again the rings were correctly identified for both datasets. This further supports the fact that the exemplars identified are of very good quality. Fast Global Kernel k-Means may be inferior to the original algorithm and the other variant when the problem is hard, as with the second dataset, but is still superior over Kernel k-Means as its clustering error is lower than the average of Kernel k-Means on this task. Finally, as expected, Kernel k-Means is very sensitive to initialization and during the 100 runs finds from near optimal solutions to very bad ones, for both datasets, making it difficult to decide on the sufficient number of restarts.

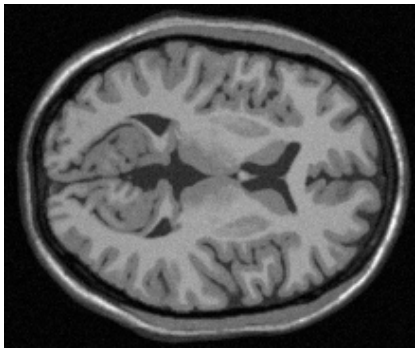
5.2 MRI Segmentation

We test Global Kernel k-Means, its two variants and Kernel k-Means with multiple restarts on simulated MRI images downloaded from the *BrainWeb* site [4]. Specifically, we use the normal brain database, which contains a single 3-d brain image, with the following simulation parameters: T1 modality, 1mm slice thickness, 3% noise and 20% intensity non-uniformity. The corresponding ground truth (discrete version) is also available and assigns to each voxel the label of the tissue that contributes most to that voxel. The tissues are divided in ten categories: background, cerebrospinal fluid (CSF), grey matter, white matter, muscle/skin, skin, skull, fat, glial matter and connective. For our experiments we focus on brain slices along the z-axis and in particular on those around the middle of the 3-d volume where the first seven tissue categories prevail, thus we consider clustering into *seven clusters*.

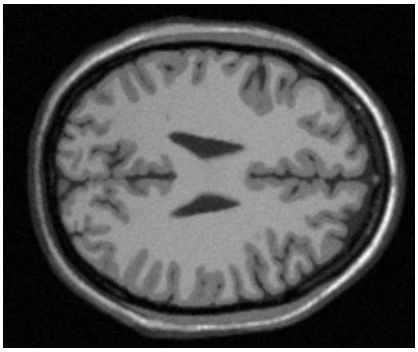
We segment slices 60, 80 and 100, shown in Figure 5.3, into seven clusters based on information derived from pixel intensities and consider only the pixels of the seven prevalent tissue categories. The corresponding ground truths are depicted in Figure 5.4. Typical approaches consider only pixel intensities as features and employ k-Means or Gaussian mixture models. In this work we suggest the use of Kernel k-Means for MRI segmentation. Moreover each pixel is represented with a vector containing not only the pixel intensity, but also the intensity histogram of a window around the pixel. The histogram is normalized so as bin quantities to represent probabilities. Based on this representation, we define a kernel (5.1) that calculates the similarity between two pixels based on the sim-



(a) Slice 60.



(b) Slice 80.



(c) Slice 100.

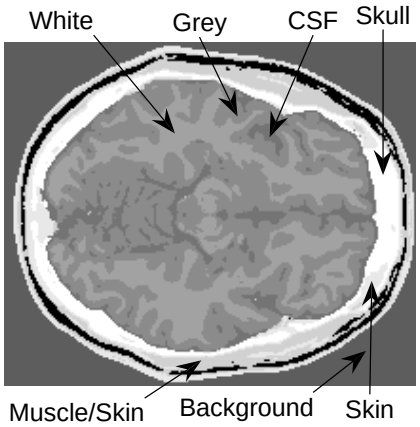
Figure 5.3: MRI slices.

ilarity between their intensities and the similarity between the corresponding histograms. More specifically, if $I(i)$ is the intensity of pixel i and $P_z(i)$ is the probability associated with the z-th bin of pixel's i histogram, the kernel element K_{ij} is computed as:

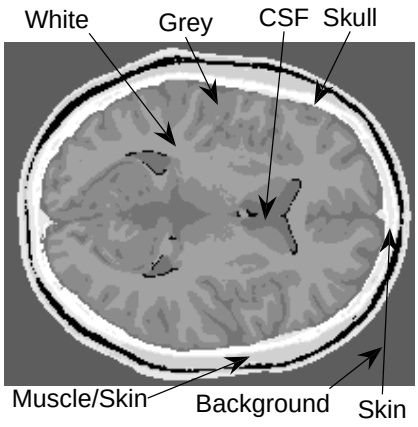
$$K_{ij} = \exp\left(\frac{-\|I(i) - I(j)\|^2}{2\sigma^2}\right)\left[\sum_{z=1}^{Bins}\sqrt{P_z(i)P_z(j)}\right] \tag{5.1}$$

The above kernel definition implies a positive semidefinite (i.e. valid) kernel matrix as it can be interpreted as a Gram matrix. The validity of the kernel is easily proven based on the property that if $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, $\mathbf{x}' = (\mathbf{x}'_a, \mathbf{x}'_b)$ and K_a, K_b are valid kernels then also $K(\mathbf{x}, \mathbf{x}') = K_a(\mathbf{x}_a, \mathbf{x}'_a)K_b(\mathbf{x}_b, \mathbf{x}'_b)$ is a valid kernel. In our case K_a is the Gaussian kernel among the intensities and K_b is simply the dot product between the square roots of the histogram vectors. Note that $K_{ii} = 1$ which is the maximum value for this kernel.

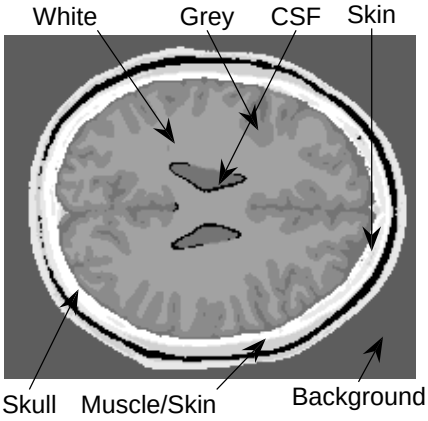
When comparing the clustering algorithms, the quality of the solution is once again measured in terms of clustering error. To decide on the kernel parameter values clustering error is not a valid measure though. Instead we define the *misclassification error* using the ground truth information. Specifically, each cluster on the final solution is assigned the label of the majority tissue class present on the cluster. The pixels of the other classes



(a) Slice 60.



(b) Slice 80.



(c) Slice 100.

Figure 5.4: Ground truth of the three slices. In black are the 3 tissues we ignore in our experiments.

in the cluster are counted as misclassifications. Summing the misclassified pixels over all clusters and dividing by the total number of pixels gives the misclassification error. By trying different combinations for the σ value, window size and number of bins we decided to use $\sigma = 0.7$, a 31-by-31 window and split the intensity range of the whole slice into 70 equally spaced bins. The above parameters were discovered using slice 80 and are applied directly on the other two slices so as to see if they are effective in nearby slices. Note that tissue distribution changes across slices.

Our experimental evaluation focuses only on the comparison of Global Kernel k-Means with CMM, Fast Global Kernel k-Means and Kernel k-Means with multiple restarts as Global Kernel k-Means incurs a very high computational cost, due to the large dataset size (each slice is of size 181-by-217), which makes its application impractical. The Global Kernel k-Means algorithm was run once, using the above parameter values, for slice 80, in order to get an estimation of how good alternatives the two speeding up schemes are for this task. The result was almost identical to that of the two variants. This is very encouraging because it seems that we can replace the original algorithm with the variants on MRI segmentation without sacrificing the quality of the solution. The original

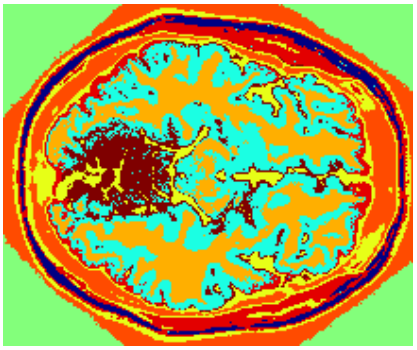
Table 5.2: Results on MRI segmentation in terms of clustering error (CE) and misclassification error (ME).

Method/Slice $\sigma = 0.7$ Win $= 31 \times 31$ Bins $= 70$		**Slice 60**		**Slice 80**		**Slice 100**	
		CE	ME	CE	ME	CE	ME
Global Kernel k-Means with CMM		5207.65	19.89%	5064.66	14.02%	5010.64	15.77%
Fast Global Kernel k-Means		5208.32	19.89%	5064.99	14.1%	5010.15	15.82%
Kernel k-Means (100 runs)	**Mean**	5286.95	19.89%	5244.39	14.01%	5094.85	15.97%
	Std	66.29		127.63		141.7	
	Min	5207.65		5064.27		5009.75	
	Max	5364.68		5477.84		5808.77	

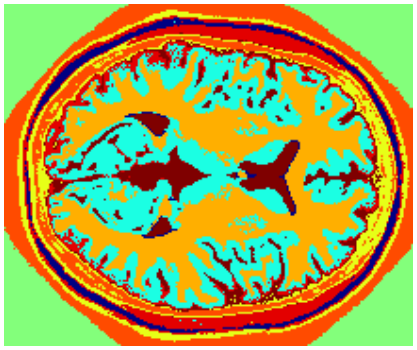
algorithm is not further considered in our experiments.

To compare the three algorithms, using the kernel given in (5.1), we run Global Kernel k-Means with CMM and Fast Global Kernel k-Means only once, while Kernel k-Means is restarted 100 times for each slice and report here the average clustering error, its standard deviation and minimum-maximum values during the 100 runs. Moreover for the Global Kernel k-Means with CMM algorithm we set $\beta = \beta_0$, $s_{ij} = \exp(-\beta(K_{ii} + K_{jj} - 2K_{ij}))$ and select the number of exemplars to be three times the number of clusters ($P = 3M$), hence we select 21 exemplars. The results for the three slices are summarized in Table 5.2 where also the misclassification error is shown. Note that for Kernel k-Means the misclassification error reported corresponds to the run with minimum clustering error. As we can see the best solutions identified by Kernel k-Means during the 100 runs have *slightly* lower clustering error than those of the two variants. For slice 60 Global Kernel k-Means with CMM achieves exactly the same error with the best run. In more detail, for slice 60, 0 and 3 out of 100 runs are better than Global Kernel k-Means with CMM and Fast Global Kernel k-Means respectively, for slice 80, 11 and 12 out of 100 runs and for slice 100, 33 and 28 out of 100 runs. The solution with minimum clustering error is located 1, 5 and 3 times during the 100 runs for slices 60, 80 and 100 respectively. A more fair and correct comparison though is the one between the average clustering error during the 100 runs and the corresponding error of the two variants where clearly Kernel k-Means is outperformed for all slices. This happens because during the restarts very bad solutions are also discovered. Moreover, because of the large dataset size executing those restarts is time consuming as the fact that Fast Global Kernel k-Means is about 20 times faster proves.

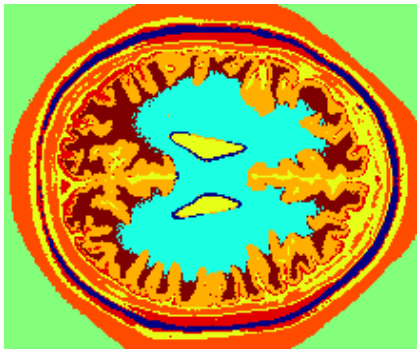
Focusing on the two variants we observe that they achieve similar clustering error. Global Kernel k-Means with CMM is better for slices 60 and 80 while Fast Global Kernel k-Means is better for slice 100. An advantage of Fast Global Kernel k-Means is its lower



(a) Slice 60.



(b) Slice 80.

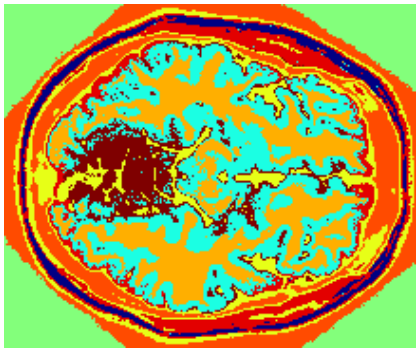


(c) Slice 100.

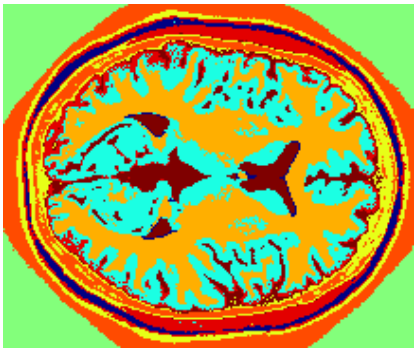
Figure 5.5: Segmented tissues of the three slices with Global Kernel k-Means with CMM. In dark blue are the 3 tissues we ignore in our experiments.

computational cost which makes him 10 times faster. The misclassification error for the three algorithms is identical for slice 60, for slice 80 Kernel k-Means is better and Global Kernel k-Means with CMM follows closely behind and finally for slice 100 Global Kernel k-Means with CMM is better with Fast Global Kernel k-Means following. By examining the results in Table 5.2 for slice 100 we observe that lower clustering error does not guarantee and lower misclassification error. We focus though primarily on clustering error when comparing the algorithms as this is the quantity optimized. All the above suggest that Fast Global Kernel k-Means is the wisest choice for MRI segmentation as it is faster than the other two methods and produces similar clustering results.

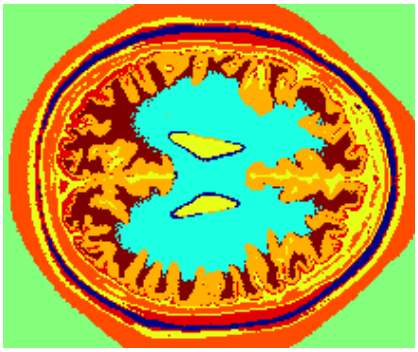
Figure 5.5 and Figure 5.6 depict the clustering result of Global Kernel k-Means with CMM and Fast Global Kernel k-Means on the three slices respectively. As we can see the clusters identified by the two algorithms are almost the same. Only a couple of pixels are placed to different clusters which are shown in Figure 5.7 in white color. Specifically, for slice 60, 242 pixels are placed to different clusters, for slice 80, only 42 pixels and for slice 100, 66 pixels. Note that slices with greater difference in clustering error have more pixels in different clusters. As the solutions of the two methods are so similar the same



(a) Slice 60.



(b) Slice 80.



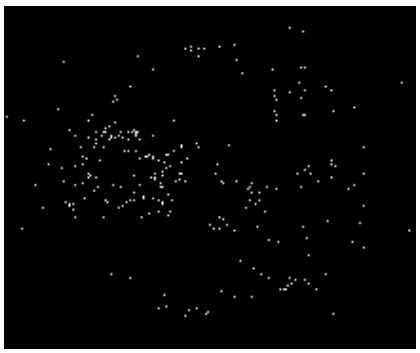
(c) Slice 100.

Figure 5.6: Segmented tissues of the three slices with Fast Global Kernel k-Means. In dark blue are the 3 tissues we ignore in our experiments.

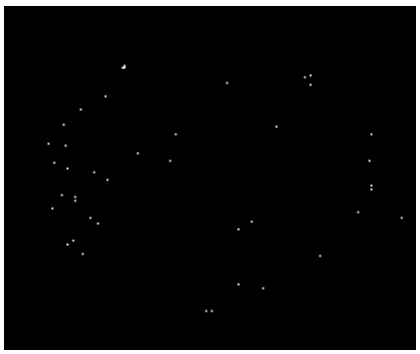
comments can be made for both.

Since different tissues share the same pixel intensities (e.g. background and skull), as shown in Figure 5.8, it is difficult to place them on different clusters even when working with a kernel that takes advantage of the window information. For slice 60 we can see that grey and white matter form pure clusters[2]. The background is split into two clusters and one of them is mixed with the skull while CSF is mixed with skin. For slice 80 grey matter, white matter, CSF and skin form quite pure clusters. Again the background is split and one part is mixed with the skull. Finally for slice 100 the result is similar to that of slice 60. Note that although some tissues are broken to two clusters, e.g. the white matter in slice 100, these clusters are pure something that is considered a good solution. Overall, the clustering performed by Global Kernel k-Means with CMM and Fast Global Kernel k-Means is satisfactory, especially for the inner part of the brain. After viewing the segmented slices it becomes even more clear that Fast Global Kernel k-Means is the

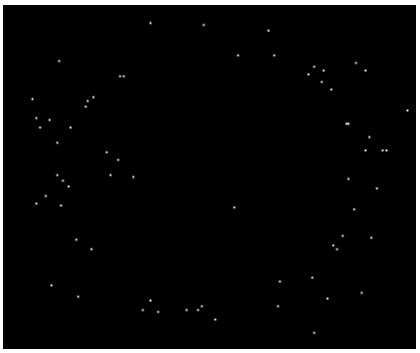
[2]By saying pure cluster we mean a cluster that mostly contains pixels of one tissue but not necessarily all the pixels of that tissue.



(a) Slice 60.



(b) Slice 80.



(c) Slice 100.

Figure 5.7: Comparison of Global Kernel k-Means with CMM and Fast Global Kernel k-Means solutions. Pixels belonging to different clusters are shown in white.

appropriate choice for MRI segmentation as it is considerably faster than the other variant and identifies similar clusters.

5.3 Pendigits Dataset

The Pendigits dataset [1] contains 7494 training digits and 3498 testing digits represented as vectors in 16-dimensional space. The features of the digits were extracted from their (x, y) coordinates, by the dataset creators. For every digit its class is also available $(0 - 9)$. We test the Global Kernel k-Means with CMM and Fast Global Kernel k-Means algorithms on the whole Pendigits dataset and also on the testing digits alone (Pendigits.tes) by performing clustering into 10 clusters. The two algorithms are compared to Kernel k-Means with multiple restarts and k-Means. We did not run the Global Kernel k-Means algorithm as the size of the dataset, 10992 digits, make its application impractical.

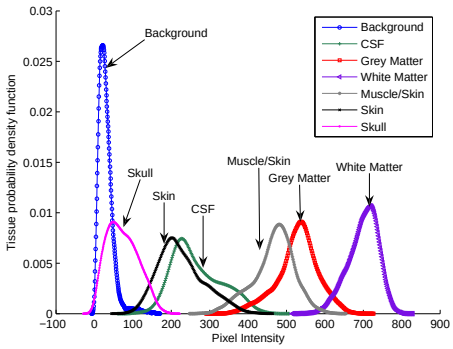


Figure 5.8: Tissue pdf estimation for slice 80.

The performance of the above algorithms is primarily measured in terms of clustering error (2.2) but as the labels of the digits is available we can form the confusion matrix between clusters and classes and calculate normalized mutual information (NMI) [7]. This measure indicates how much the clustering and true class labels match and is defined as:

$$NMI = \frac{2\sum_{l=1}^{M}\sum_{h=1}^{c}\frac{n_l^h}{N}\log\frac{n_l^h N}{\sum_{i=1}^{M} n_i^h \sum_{i=1}^{c} n_l^i}}{\mathbb{H}(\pi)+\mathbb{H}(\zeta)} \tag{5.2}$$

where N is the dataset size, M is the number of clusters, c is the number of classes, n_l^h is the number of points in cluster l from class h, $\mathbb{H}(\pi) = -\sum_{i=1}^{M}\frac{n_i}{N}\log\frac{n_i}{N}$ is the entropy of the clusters and $\mathbb{H}(\zeta) = -\sum_{i=1}^{c}\frac{n^i}{N}\log\frac{n^i}{N}$ is the entropy of the classes. High NMI values indicate that clustering and true class labels match well.

For our experiments we once again use the Gaussian kernel and the same σ value is used in all algorithms in order to obtain a meaningful comparison. Note that k-Means performs clustering in input space and there is no need to use a kernel function. For the Global Kernel k-Means with CMM algorithm we set $\beta = \beta_0$, $s_{ij} = \exp(-\beta\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2)$ and select twice as many exemplars as the number of clusters ($P = 2M$), hence we select 20 exemplars. This algorithm and Fast Global Kernel k-Means are run only once. Kernel k-Means and k-Means are run 100 times and report here the average and minimum performance values during the 100 runs. The results are shown in Table 5.3. Note that the NMI values reported on the '**Min**' rows are those achieved by the restart with minimum clustering error.

In Table 5.3 we do not report the clustering error for k-Means as this algorithm performs clustering in input space while the others in feature space, so the values are not comparable. The NMI values of k-Means are inferior to those of the other algorithms and thus back our decision to apply feature space clustering for the Pendigits dataset. When clustering the whole Pendigits dataset we observe that Fast Global Kernel k-Means, Global Kernel k-Means with CMM and the run of Kernel k-Means with minimum clustering error are evenly matched. The solution with minimum clustering error is identified in only 7 out of 100 runs by Kernel k-Means. The average clustering error and NMI values though are considerably worse as very bad solutions are identified during the restarts.

Table 5.3: Pendigits results in terms of clustering error (CE) and normalized mutual information (NMI).

Method/Dataset		**Pendigits Full** $\sigma = 2.1$		**Pendigits.tes** $\sigma = 2.8$	
		CE	NMI	CE	NMI
Global Kernel k-Means with CMM		6514.95	0.776	1490.44	0.749
Fast Global Kernel k-Means		6514.95	0.776	1504.81	0.75
Kernel k-Means (100 runs)	**Mean**	6668.34	0.739	1537.69	0.713
	Min	6514.94	0.777	1485.2	0.754
k-Means (100 runs)	**Mean**	–	0.688	–	0.685
	Min	–	0.697	–	0.701

When considering only the testing digits, Kernel k-Means achieves lower clustering error on its best run but on average it is worse than the other two algorithms. In more detail, 15 out of 100 runs achieve lower clustering error than Fast Global Kernel k-Means but only 3 out of 100 achieve lower than Global Kernel k-Means with CMM. As Global Kernel k-Means with CMM has lower clustering error than Fast Global Kernel k-Means and is very close to the best run of Kernel k-Means we can state that the exemplars identified are of good quality. As for the NMI values they are similar for the three algorithms, despite the differences in clustering error, except from the average NMI of Kernel k-Means which is quite lower due to bad solutions occurring during the restarts.

5.4 Olivetti Dataset

The Olivetti face database [21] contains ten 64×64 gray scale images of each of 40 individuals. We selected the same subset of images and applied the same preprocessing as in [11]. Specifically, we selected the first 100 images, belonging to 10 individuals, and smoothed them with a Gaussian kernel with $\sigma = 0.5$. In order to obtain more variations for each image we applied 3 in-plane rotations ($-10°$, $0°$ and $10°$) and 3 scalings (0.9, 1.0 and 1.1) thus producing a dataset with 900 images in total. To avoid including the background behind each face we extracted a central 50×50 window and finally the pixel intensities of each image were normalized to zero mean and 0.1 standard deviation.

We cluster images using the standard optimization criterion of squared error but on feature space instead of input space. We test the Global Kernel k-Means, Fast Global Kernel k-Means and Global Kernel k-Means with CMM algorithms on the 900 images and compare them to Kernel k-Means with multiple restarts as well as affinity propagation [11]. Affinity propagation was tested on this dataset in [10, 11] achieving good results and

thus serves as a good measure for the performance of our methods. For our experiments, each image is represented with a 2500-dimensional vector containing the pixel intensities. Again we use the Gaussian kernel and the same σ for all algorithms. For the Global Kernel k-Means with CMM algorithm we set $\beta = \beta_0$, $s_{ij} = \exp(-\beta\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2)$ and the number of selected exemplars equals three times the number of clusters ($P = 3M$). For affinity propagation we set the preferences equal to a common value that yields the desired number of clusters and the pairwise similarities equal to $s(i,j) = -\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|^2$. Finally Kernel k-Means is restarted 100 times.

We partition the dataset in different number of clusters using all the above algorithms and for every experiment we report the clustering error (2.2) and the misclassification error defined in section 5.2. For the misclassification error each cluster is associated with the individual whose images are the majority in that cluster. Images belonging to other individuals are counted as misclassifications. As the number of clusters increases the misclassification error naturally drops.

Figure 5.9 shows the clustering error achieved by the algorithms for different number of clusters. We do not report the clustering error for affinity propagation as it is naturally much higher relative to the other algorithms, since affinity propagation identifies exemplars as the centers of the clusters. As we can see the performance of the algorithms is very similar for 10 clusters but as the number of clusters increases and the problem gets harder the differences in performance become clear. Global Kernel k-Means is the best performer achieving constantly the lowest clustering error. This once again demonstrates that, due to its exhaustive nature, this method identifies near optimal solutions. Global Kernel k-Means with CMM is the second best algorithm for 20 or more clusters. This backs our claim that good exemplars are identified and then fine tuned with Global Kernel k-Means. Fast Global Kernel k-Means is very close to the best run of Kernel k-Means, a little worse for 50 or less clusters and better for more. Finally, the average clustering error of Kernel k-Means is the highest in all cases demonstrating that bad solutions are identified during the restarts.

In Figure 5.10 the algorithms are compared in terms of misclassification error. Their behavior is not the same and as smooth as with clustering error, leading to the conclusion that lower clustering error does not necessarily imply and lower misclassification error. There are some similarities though as Kernel k-Means gradually becomes the worst performer as the clusters increase, except some fluctuations for 50 and 150 clusters, and also Global Kernel k-Means is among the best algorithms in all cases. Affinity propagation starts as the worst algorithm and ends as the best. The two variants of Global Kernel k-Means are inferior to the original algorithm but better from Kernel k-Means for 50 and 70 clusters.

Overall, we can state that Global Kernel k-Means is the best algorithm followed by the CMM variant, the Fast Global Kernel k-Means algorithm and finally Kernel k-Means. This observation is primarily based on clustering error performance as that is the objective optimized by the algorithms. As for affinity propagation its performance appears to be

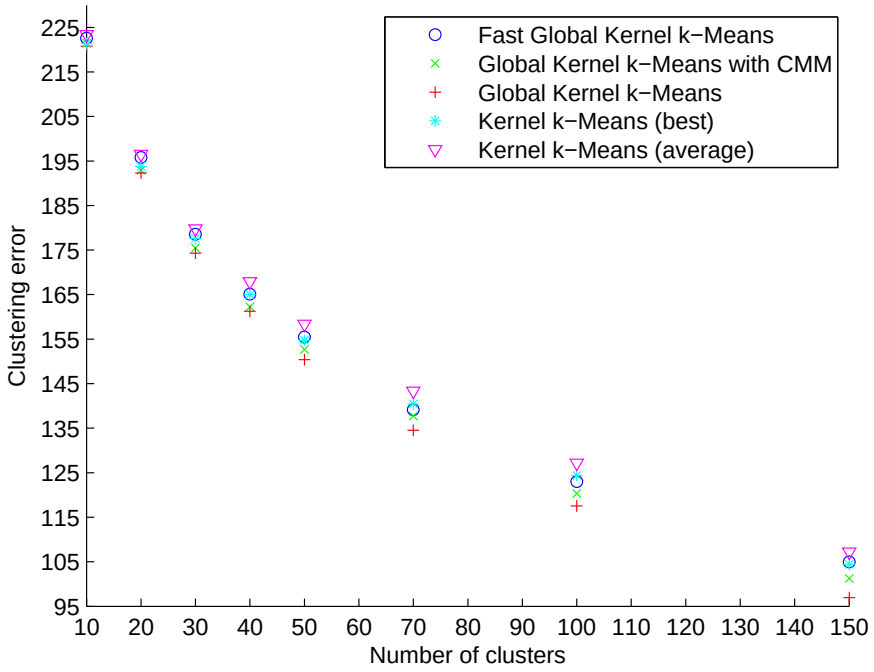


Figure 5.9: A comparison of the clustering error achieved by Global Kernel k-Means and its variants as well as Kernel k-Means on the Olivetti face database.

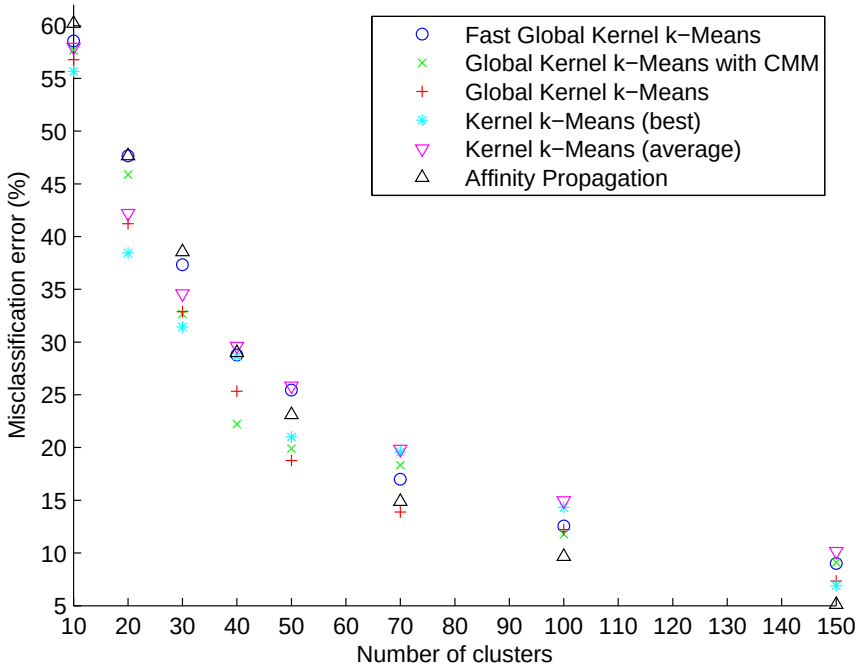


Figure 5.10: A comparison of the misclassification error achieved by Global Kernel k-Means and its variants as well as Kernel k-Means and affinity propagation on the Olivetti face database.

Table 5.4: Test graphs.

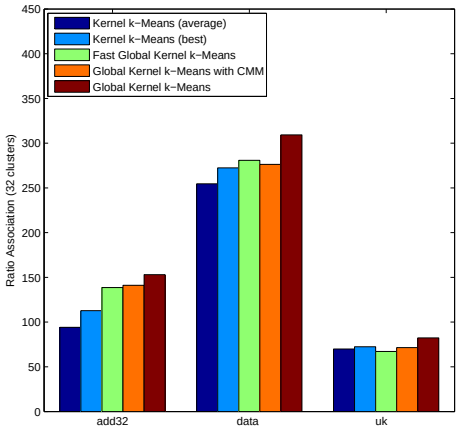
Graph Name	Number of Nodes	Number of Edges
add32	4960	9462
data	2851	15093
uk	4824	6837

close to that of Global Kernel k-Means for 70 or more clusters but worse for fewer clusters (Figure 5.10).

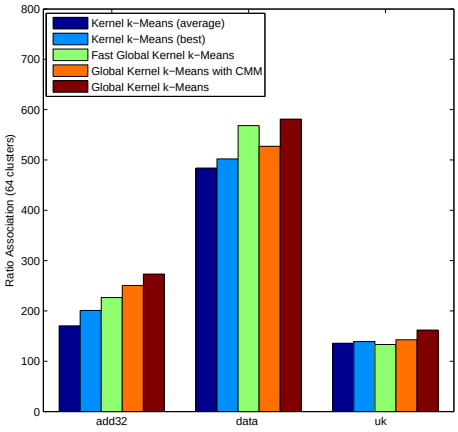
5.5 Graph Partitioning

As already discussed the weighted version of Kernel k-Means can be used to optimize many graph cut objectives. As Kernel k-Means lies in the heart of the Global Kernel k-Means algorithm and its two variants, we have seen that weights can be readily applied to these algorithms so they can also be used for graph partitioning. In our experiments we test Global Kernel k-Means and its two variants against Kernel k-Means with multiple restarts on the three graphs listed in Table 5.4 and made available through the graph partitioning archive [24]. These graphs are undirected and all edge weights are equal to one so the affinity matrix is symmetric and its entries are either zero or one. Each graph is partitioned into 32, 64 and 128 clusters and for each number of clusters we maximize the ratio association objective and we also minimize the normalized cut objective. The kernel and the weights are defined as discussed in section 4.3. A problem that usually occurs is that the kernel matrix might not be positive semidefinite and thus the algorithms may not converge. For this reason we diagonally shift the kernel matrix by a small amount as mentioned in section 4.3. Note that we perform a diagonal shift that does not necessarily make the kernel matrix positive semidefinite but is sufficient for the algorithms to converge. This is done in order to avoid adding a big diagonal shift which consequently will decrease the performance of the algorithms as shown in [8]. For the Global Kernel k-Means with CMM algorithm we set $\beta = \beta_0$, $s_{ij} = \exp(-\beta(K_{ii} + K_{jj} - 2K_{ij}))$ and select twice as many exemplars as the number of clusters ($P = 2M$). Finally Kernel k-Means is restarted 50 times.

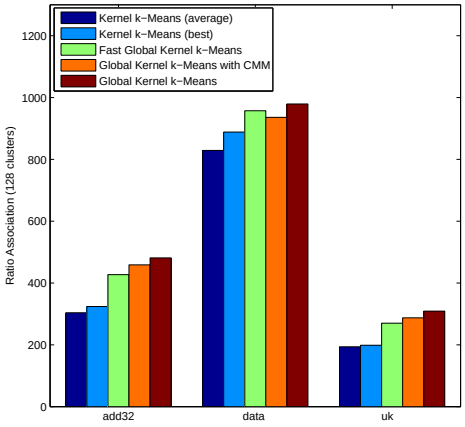
Figure 5.11 depicts the ratio association values achieved by the algorithms for 32, 64 and 128 clusters on the graphs listed in Table 5.4. For this task we use the unweighted version of the algorithms as $w_i = 1$. Clearly Global Kernel k-Means is the best algorithm as it achieves the highest ratio association for all graphs and all number of clusters. For the *add32* and *data* graphs its two variants are also better than the best run of Kernel k-Means. Surprisingly, for the *data* graph Fast Global Kernel k-Means is superior to Global Kernel k-Means with CMM. For the *uk* graph and 32 clusters the two variants



(a) 32 clusters.



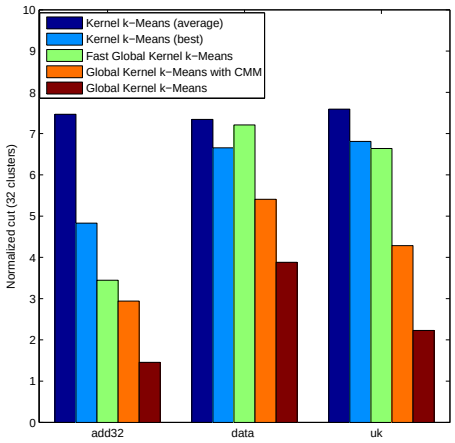
(b) 64 clusters.



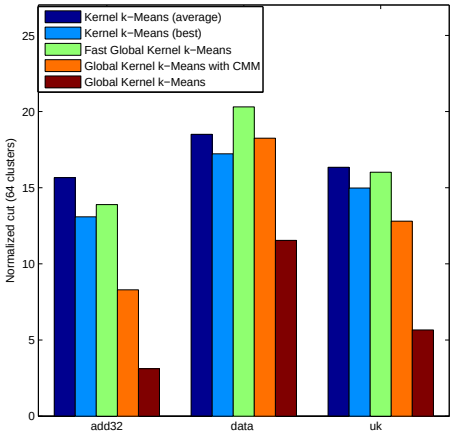
(c) 128 clusters.

Figure 5.11: Ratio association values achieved by Global Kernel k-Means and its variants as well as Kernel k-Means.

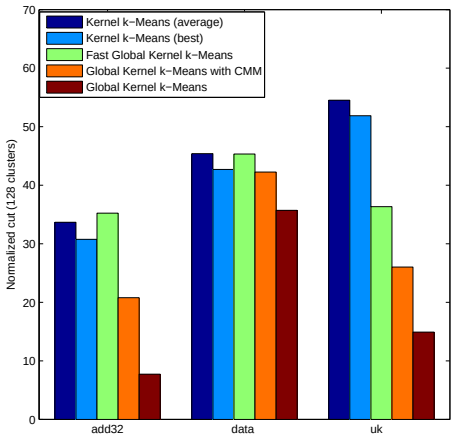
are inferior to the best run of Kernel k-Means, but as the number of clusters increases and the problem gets harder, first Global Kernel k-Means with CMM jumps ahead for 64 clusters and then Fast Global Kernel k-Means follows for 128 clusters. For this graph and 128 clusters the ratio association achieved by the two variants is 38% higher than that of the best run of Kernel k-Means while for *add32* graph and 128 clusters it is 36% higher. This shows that the speeding up schemes are considerbaly better than Kernel k-Means in many cases. Also note that the two variants are very close to the original algorithm in some experiments, such as *data* graph for 64 and 128 clusters where the original algorithm achieves only 2% higher ratio association than the best variant and *add32* graph for 128 clusters where it achieves 5% higher ratio association than the best variant. This demonstrates that there are cases where the two speeding up schemes lower the computational complexity without degrading the quality of the solution of the original



(a) 32 clusters.



(b) 64 clusters.



(c) 128 clusters.

Figure 5.12: Normalized cut values achieved by Global Kernel k-Means and its variants as well as Kernel k-Means.

algorithm. Between the two variants there is no clear winner as Global Kernel k-Means with CMM is better for the *add32* and *uk* graphs while Fast Global Kernel k-Means is superior for the *data* graph.

Results on normalized cut values for 32, 64 and 128 clusters are shown in Figure 5.12. For this task we use the weighted version of the algorithms described in chapter 4. In these experiments we do not set $\beta = \beta_0$ for the Global Kernel k-Means with CMM algorithm as the empirical value β_0 is huge, which combined with the fact that the kernel matrix is not positive semidefinite results in bad exemplars. Specifically, we set $\beta = 10^{-3}\beta_0$, $\beta = 10^{-5}\beta_0$ and $\beta = 10^{-2}\beta_0$ for *add32*, *data* and *uk* graphs respectively. We do not set β equal to the same fraction of β_0 for all graphs as our study and experiments indicated that the fraction of β_0 where good exemplars occur differs for each graph. Once again Global Kernel k-Means is the best performer for all graphs and all number of clusters as

it achieves the lowest normalized cut which in some cases is 2 to 3 times less than that of the second best algorithm. Global Kernel k-Means with CMM, although not close to the original algorithm this time, is clearly the second best method as only on the *data* graph for 64 clusters it is beaten by the best run of Kernel k-Means. Based on this, we can safely state that the β values as defined above result in good exemplars. In general, Fast Global Kernel k-Means has a similar performance to the best run of Kernel k-Means. There are cases where Fast Global Kernel k-Means is better, such as *add32* graph for 32 clusters and *uk* graph for 32 and 128 clusters, and others where Kernel k-Means gets ahead, such as *data* graph for 64 clusters and *add32* graph for 128 clusters.

Overall, we have seen that Global Kernel k-Means can be effectively applied to graph partitioning as it clearly outperforms Kernel k-Means with multiple restarts on this task and also is the best algorithm of those considered in all experiments. The two variants are very good alternatives to the original algorithm especially when the ratio association criterion is considered. The non random choices of the original algorithm and its two speeding up schemes boost the performance considerably, thus making them a very good choice for graph partitioning. For the normalized cut criterion, although Global Kernel k-Means with CMM is inferior to the original algorithm, we may still consider its application in place of Global Kernel k-Means, if lower computation time is an important issue, as it is the second best algorithm. Unfortunately, the same cannot be said for Fast Global Kernel k-Means as it is very close to Kernel k-Means with multiple random restarts and outperformed by Global Kernel k-Means with CMM on this task.

Chapter 6

Conclusions and Future Work

We have proposed the Global Kernel k-Means clustering algorithm, a method that maps data points from input space to a higher dimensional feature space through the use of a kernel function and optimizes the clustering error in the feature space by locating near optimal solutions. The main advantages of this method are its deterministic nature, which makes it independent of cluster initialization, and the ability to identify nonlinearly separable clusters in input space. Another important feature of the proposed algorithm is that in order to solve the M-clustering problem, all intermediate clustering problems with $1, \ldots, M$ clusters are solved. This may prove useful in problems where we seek the true number of clusters.

Moreover, we developed two variants of the algorithm to accelerate its execution. The Fast Global Kernel k-Means variant considerably reduces the computational cost by requiring one run of Kernel k-Means for each intermediate problem. The Global Kernel k-Means with CMM variant, first identifies a number of good exemplars, by fitting a convex mixture model to the data, and then tries only these exemplars as possible initializations for the newly added cluster in each of the intermediate problems. This variant has less computational savings, but it provides solutions closer to those of the original algorithm as shown by our experiments.

We also extended the above algorithms to handle weighted data points based on the weighted Kernel k-Means algorithm. The use of weights makes possible the application of these methods to graph partitioning as their objective function becomes equivalent to that of many graph cut criteria if the weights and kernel matrix are set appropriately.

The aforementioned methods have been tested on many datasets in order to ensure their broad applicability and draw reliable conclusions. In general, we could state that Global Kernel k-Means and its two variants outperform Kernel k-Means with multiple restarts. Global Kernel k-Means, whenever its application is possible, is the best performer. For the two large datasets, MRI images and Pendigits, where the computational cost makes the application of this algorithm prohibitive, the two variants are very good alternatives and, in particular, Fast Global Kernel k-Means for MRI segmentation and Global Kernel k-Means with CMM for handwritten digits. For the graph partitioning

problem where the weighted versions are used, the Global Kernel k-Means algorithm is by far the best and the two variants are good alternatives when the ratio association is optimized. For normalized cut only the Global Kernel k-Means with CMM variant provides solutions similar to the original algorithm. Overall we conclude that whenever the dataset size allows the use of the exact Global Kernel k-Means method then it is the best choice. If the dataset is large or time is a critical factor then the Global Kernel k-Means with CMM variant could be used instead and if further acceleration is required the Fast Global Kernel k-Means variant could be employed.

As for future work, a possible direction is the use of parallel processing to accelerate the Global Kernel k-Means algorithm since the local search performed when solving the k-clustering problem requires running Kernel k-Means N times and these executions are independent of each other. Another important issue is the development of theoretical results concerning the near-optimality of the obtained solutions. Also we plan to use Global Kernel k-Means in conjunction with criteria and techniques for estimating the optimal number of clusters. The integration of the Global Kernel k-Means algorithm with other exemplar-based techniques is another possible direction for future work. Finally, the application of this algorithm to graph partitioning needs further investigation and a comparison with spectral methods and other graph clustering techniques is required.

Bibliography

[1] A. Asuncion and D. Newman. UCI machine learning repository, 2007. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[2] A. M. Bagirov. Modified global k-means algorithm for sum-of-squares clustering problems. *Pattern Recognition*, 41(10):3192–3199, October 2008.

[3] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[4] C. A. Cocosco, V. Kollokian, R. K.-S. Kwan, and A. C. Evans. Brainweb: Online interface to a 3-d MRI simulated brain database. *NeuroImage*, 5(4), 1997. [Online]. Available: http://www.bic.mni.mcgill.ca/brainweb/.

[5] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, 1991.

[6] I. Csiszár and P. C. Shields. Information theory and statistics: A tutorial. *Communications and Information Theory*, 1(4):417–528, 2004.

[7] I. S. Dhillon, Y. Guan, and B. Kulis. Kernel k-means, spectral clustering and normalized cuts. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 551–556, 2004.

[8] I. S. Dhillon, Y. Guan, and B. Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, University of Texas at Austin, 2004.

[9] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, November 2007.

[10] D. Dueck and B. Frey. Non-metric affinity propagation for unsupervised image categorization. In *Proceedings of the 11th IEEE International Conference on Computer Vision*, pages 1–8, October 2007.

[11] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315:972–976, 2007.

[12] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[13] R. Hathaway, J. Bezdek, and Y. Hu. Generalized fuzzy c-means clustering strategies using L_p norm distances. *IEEE Transactions on Fuzzy Systems*, 8(5):576–582, October 2000.

[14] F. Höppner, F. Klawonn, R. Kruse, and T. Runkler. *Fuzzy cluster analysis: Methods for classification, data analysis and image recognition*. Wiley, 1999.

[15] J. Kim, K.-H. Shim, and S. Choi. Soft geodesic kernel k-means. In *Proceedings of the 32nd IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2, pages 429–432, 2007.

[16] D. Lashkari and P. Golland. Convex clustering with exemplar-based models. In *Advances in Neural Information Processing Systems 20*, pages 825–832, 2008.

[17] J. Li, D. Tao, W. Hu, and X. Li. Kernel principle component analysis in pixels clustering. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 786–789, September 2005.

[18] A. Likas, N. Vlassis, and J. J. Verbeek. The global k-means clustering algorithm. *Pattern Recognition*, 36(2):451–461, February 2003.

[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*, pages 281–297, 1967.

[20] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2002.

[21] Olivetti and Oracle Research Laboratory. The olivetti & oracle research laboratory database of faces, 1994.

[22] B. Scholkopf, A. J. Smola, and K.-R. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, July 1998.

[23] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.

[24] C. Walshaw. The graph partitioning archive, 2007. [Online]. Available: http://staffweb.cms.gre.ac.uk/∼ c.walshaw/partition/.

[25] Y. Weiss. Segmentation using eigenvectors: A unifying view. In *Proceedings of the 7th IEEE International Conference on Computer Vision*, pages 975–982, 1999.

[26] R. Xu and D. Wunsch II. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[27] S. X. Yu and J. Shi. Multiclass spectral clustering. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 313–319, 2003.

[28] H. Zha, X. He, C. H. Q. Ding, M. Gu, and H. D. Simon. Spectral relaxation for k-means clustering. In *Advances in Neural Information Processing Systems 14*, pages 1057–1064, 2002.

[29] R. Zhang and A. I. Rudnicky. A large scale clustering scheme for kernel k-means. In *Proceedings of the 16th International Conference on Pattern Recognition*, pages 289–292, 2002.

[30] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 103–114, 1996.

Author's Publications

1. G. Tzortzis and A. Likas. The global kernel k-means clustering algorithm. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1978-1985, 2008.

2. G. Tzortzis and A. Likas. Deep belief networks for spam filtering. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*, volume 2, pages 306-309, 2007.

Short Vita

Grigorios Tzortzis was born in Ioannina, Greece in 1984. He graduated from the Dodonaia Ekpaideutiria high school in Ioannina in 2002 and at the same year he was admitted at the Computer Science Department of the University of Ioannina. He received his BSc degree in computer science in 2006 with degree "excellent". That year he became a postgraduate student at the same institution from where he graduated in September 2008. His main research interests lie in the field of machine learning and data mining, with specific interests in clustering and text classification.