KERNEL PROBABILISTIC DISTANCE CLUSTERING ALGORITHMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

DİLAY ÖZKAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JULY 2022

Approval of the thesis:

KERNEL PROBABILISTIC DISTANCE CLUSTERING ALGORITHMS

submitted by **DİLAY ÖZKAN** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Esra Karasakal
Head of Department, **Industrial Engineering** _____

Prof. Dr. Cem İyigün
Supervisor, **Industrial Engineering, METU** _____

Examining Committee Members:

Prof. Dr. Sinan Gürel
Industrial Engineering, METU _____

Prof. Dr. Cem İyigün
Industrial Engineering, METU _____

Prof. Dr. Yasemin Serin
Industrial Engineering, METU _____

Assist. Prof. Dr. Sakine Batun
Industrial Engineering, METU _____

Assist. Prof. Dr. Fatma Yerlikaya Özkurt
Industrial Engineering, Atılım University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: DİLAY ÖZKAN

Signature :

ABSTRACT

KERNEL PROBABILISTIC DISTANCE CLUSTERING ALGORITHMS

ÖZKAN, DİLAY
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Cem İyigün

July 2022, 100 pages

Clustering is an unsupervised learning method that groups data considering the similarities between objects (data points). Probabilistic Distance Clustering (PDC) is a soft clustering approach based on some principles. Instead of directly assigning an object to a cluster, it assigns them to clusters with a membership probability. PDC is a simple yet effective clustering algorithm that performs well on spherical-shaped and linearly separable data sets.

Traditional clustering algorithms fail when the data set is non-spherical or non-linearly separable, as in the case of PDC. The kernel method overcomes this problem by implicitly mapping the data into a higher dimensional space via a nonlinear transformation, where the data may be linearly separable. This study focuses on developing kernel-based clustering algorithms using the principles of PDC to overcome the problem of clustering non-spherical or non-linearly separable data sets and proposes three kernel-based PDC algorithms. In addition, different than the classical approach, Mahalanobis distance is also considered in kernel clustering, and a new kernel-based Mahalanobis distance is developed to be used in soft kernel clustering techniques. An experimental study is conducted for real and synthetic data sets to

measure the performance of the proposed kernel-based PDC algorithms.

Keywords: clustering, probabilistic clustering, kernel functions, kernel method

ÖZ

ÇEKİRDEK OLASILIKSAL MESAFE KÜMELEME ALGORİTMALARI

ÖZKAN, DİLAY
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Cem İyigün

Temmuz 2022 , 100 sayfa

Kümeleme, nesneler (veri noktaları) arasındaki benzerlikleri dikkate alarak verileri gruplayan gözetimsiz bir öğrenme yöntemidir. Olasılıksal Mesafe Kümelemesi (PDC), bazı ilkelere dayanan yumuşak bir kümeleme yaklaşımıdır. Bir nesneyi doğrudan bir kümeye atamak yerine, bunları üyelik olasılığı ile kümelere atar. PDC, küresel şekilli ve doğrusal olarak ayrılabilir veri kümelerinde iyi performans gösteren basit ama etkili bir kümeleme algoritmasıdır.

Geleneksel kümeleme algoritmaları, PDC durumunda olduğu gibi, veri setleri küresel olmadığında veya doğrusal olarak ayrılamadığında başarısız olur. Çekirdek yöntemi, verilerin doğrusal olarak ayrılabilir olabileceği, doğrusal olmayan bir dönüşüm yoluyla verileri dolaylı olarak daha yüksek boyutlu bir uzaya eşleyerek bu sorunun üstesinden gelir. Bu çalışma, küresel olmayan veya doğrusal olarak ayrılamayan veri setlerinin kümelenmesi sorununun üstesinden gelmek için PDC ilkelerini kullanan çekirdek tabanlı kümeleme algoritmaları geliştirmeye odaklanır ve üç çekirdek tabanlı PDC algoritması önerir. Ayrıca, çekirdek kümelemede klasik yaklaşımdan farklı olarak Mahalanobis mesafesi de dikkate alınmış ve yumuşak çekirdek küme-

leme tekniklerinde kullanılmak üzere yeni bir çekirdek tabanlı Mahalanobis mesafesi geliştirilmiştir. Önerilen çekirdek tabanlı PDC algoritmalarının performansını ölçmek için gerçek ve sentetik veri setleri için deneysel bir çalışma yapılmıştır.

Anahtar Kelimeler: kümeleme, olasılıksal kümeleme, çekirdek fonksiyonları, çekirdek yöntemi

To my tears ...

ACKNOWLEDGMENTS

I want to express my endless thanks to my thesis advisor, Cem Iyigun. He has been more than an advisor to me; I have learned much from him about academia and life. I am eternally grateful to him for giving me the honor of being his student. I am fortunate to have an advisor and mentor like him.

I want to thank the jury members of the thesis examining committee, Prof. Dr. Sinan Gürel, Prof. Dr. Yasemin Serin, Assist. Prof. Dr. Sakine Batun, and Assist. Prof. Dr. Fatma Yerlikaya Özkurt for their precious comments and feedback.

I would like to thank my dear mother, Selda, for giving great importance to my education and believing that I will always be successful throughout my life. A big thank you to my father, Emin, for showing me that I can achieve anything by working hard. I want to thank my sister Damla, the best sister in the world, for enduring me with good and bad and being my best friend. Finally, special thanks to my dear brother Mehmet Emin for reminding me to always believe in myself.

I acknowledge my good-hearted and perfect friends Beril Akkaya and Can Er for always being there for me in both good and bad times. I would like to express my gratitude to my friends Yağmur Caner, Erdinç Durak, and Tuna Berk Kaya for always understanding me. In addition, I would like to thank my friend and colleague Alper Şener for his support.

I cannot thank Ceyhan enough for never leaving me alone and walking this path with me. I will always be grateful to him for being the savior of my bad days, the sunshine of my good days, and a part of my inspiration.

I owe an acknowledgment to the faculty members of the Industrial Engineering Department of METU. Finally, I would like to thank TÜBİTAK for financially supporting me through the 2210-A scholarship program during my graduate education.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ALGORITHMS

ALGORITHMS

LIST OF ABBREVIATIONS

FCM	Fuzzy c-Means
GG	Gath-Geva
GK	Gustafson Kessel
KPD	Kernel Probabilistic Distance Clustering
KPD-C	Kernel Probabilistic Distance Clustering with Cluster Centers
KPD-F	Kernel Probabilistic Distance Clustering in Feature Space
KPD-M	Kernel Pd-clustering with Mahalanobis Distance
PDC	Probabilistic D-Clustering
PDQ	Probabilistic DQ-Clustering
SVM	Support Vector Machines

CHAPTER 1

INTRODUCTION

Machine learning is a process of extracting the hidden patterns from the data [1]. It is one of the steps of knowledge discovery process, where the aim is to discover useful knowledge from the data. Machine learning contains several approaches, and they can be divided mainly and classically into two categories based on the information provided by the data: *supervised learning* and *unsupervised learning*.

Supervised learning aims at finding a pattern using the data with information (label), and then developing a model to predict the outcome of unlabeled data. The goal of unsupervised learning, on the other hand, is to extract information from the data with no labels. The most common method of unsupervised learning is *clustering*. It is a process of exploring groups in the data based on the similarity between the data points (objects). The objective of clustering is grouping the similar objects in the same cluster while separating the dissimilar objects from each other.

Clustering methods can be divided into two categories as *hierarchical* and *partitional clustering*. Hierarchical clustering is a non-parametric approach that aims at constructing a hierarchical structure of clusters based on a distance measure. Partitional clustering represents each cluster with a cluster prototype (i.e. center, median, etc.). It constructs clusters by considering optimization of an objective function and the proximity between objects (data points) and prototypes. Partitional clustering is split into two based on the assignment types. Hard (crisp) assignment methods assign each object to exactly one cluster; therefore, it comes up with disjoint clusters. Soft assignment methods, on the other hand, assigns objects to clusters with a probability (membership degree) instead of directly assigning them to only one cluster.

Partitional clusters create hypersurfaces among clusters. According to those hypersurfaces, clusters in a data set are determined. Although there is no prior information, the shape or the distribution of the clusters in a data set is important for a success of a clustering task. In some cases, points in a data set cannot be linearly separable, i.e. a hypersurface cannot separate the data. For those data sets, nonlinear separating hypersurfaces between clusters should be constructed. Classical partitional clustering approaches do not perform well in such data sets.

Kernel clustering methods aim to group non-linearly separable data sets, and they create nonlinear hypersurfaces for clustering. Kernel methods implicitly map the data from the original space (input space) into a higher dimensional space (feature space), where the data is assumed to be linearly separable in that space, and then group the data set using linear partitioning method. Kernel methods allow us to cluster the data in the feature space without obtaining the mapped data points explicitly. To do so, the mapping functions that map the data set into a higher dimensional space should satisfy some properties. Thus, calculations in the feature space can be done using the data set in the original space using a similarity function called *kernel function*. Kernel functions provide the inner product of mapped data points in the feature space by taking the data points in the original space as input. This property of kernel functions reduces the computational complexity.

This thesis focuses on a soft clustering approach, Probabilistic Distance Clustering (PDC) [2], and it works on the kernelization of this clustering approach. Rather than using the squared distance function, PDC works with distance itself. Therefore, it is insensitive in noise environment. PDC works well on spherical-shaped data sets. However, it does not give good results on data sets with different shapes, such as non-spherical or non-linearly separable data sets. In addition, it assumes that the cluster sizes are fixed and equal.

1.1 Contribution to the Literature

To overcome the drawbacks of PDC, we adapt kernel methods to PDC. *Kernel-based PDC approach* has not been studied in the literature. Using the principles

of PDC algorithm, kernel version of the algorithm is studied. We propose three kernel-based PDC algorithms. The first algorithm, named as KPD later, is a novel approach when comparing with kernel-based clustering algorithms in the literature. The second algorithm, called KPD-F, is another kernel approach where it handles the center representation and its calculations in the original space, different than the first approach. These two algorithms use Euclidean distance as a distance metric. In the third algorithm, we focus on using the Mahalanobis distance and propose two kernelized Mahalanobis distance functions. Then these distance functions are implemented into the kernel clustering, and we develop a new kernel-based PDC with Mahalanobis distance, named as KPD-M. Different than the literature, this new approach implements Mahalanobis distance for kernel method without regularizing the objective function of the clustering problem. With an extensive computational experiments, the success of the proposed kernel methods is demonstrated by comparing with the state-of-the-art soft kernel clustering methods.

The thesis is organized as follows. Chapter 2 provides a background on clustering, literature of soft clustering approaches, and kernels. Literature review on kernel clustering methods is given in Chapter 3. In Chapter 4, we introduce the principles of kernel-based PDC approach and propose KPD and KPD-F algorithms. Kernelized Mahalanobis distance and the novel kernel Mahalanobis PDC algorithm, KPD-M, is provided in Chapter 5. Chapter 6 contains the experimental results of three algorithms on both real and synthetic data sets, and compares them with those of kernel-based soft clustering approaches. Finally, Chapter 7 concludes this study and refers to the future research directions.

CHAPTER 2

A BRIEF BACKGROUND ON CLUSTERING AND KERNELS

Clustering is an unsupervised learning method where patterns do not have any label that shows the group they belong to. The aim of clustering is to group the patterns (or objects) such that each group contains similar data points. In addition, groups are created such that each group should be dissimilar to each other. Therefore, the decision of how to measure the similarity gains importance.

Distance between the objects is one of the commonly used similarity measures in clustering. The smaller the distance between two objects, the greater the similarity of them. Similarity is defined based on the type of object attributes. Features can be divided into two groups as *continuous attributes* and *categorical attributes*. Similarity measure for each type varies. For the objects having the former type attributes, common distance functions can be listed as Manhattan distance, Euclidean distance, and Mahalanobis distance. Jaccard index and Hamming distance are commonly used similarity measures for the objects with categorical attributes.

Besides determining a similarity measure, deciding on the objective of clustering is important. One of the aims of clustering is to construct clusters such that they are separated from each other. That is, clusters should be as much as dissimilar to each other. This approach is called *separation*. In addition, similar objects should be grouped in the same cluster, i.e. *compactness* should be maximized.

There exists various clustering algorithms in the literature. [3] grouped those algorithms into two as *hierarchical clustering* and *partitional clustering*.

2.1 Hierarchical Clustering

Hierarchical clustering is a nested clustering approach where each cluster is a subset of a larger cluster. This method only takes the data as input and returns with a dendrogram, a structure that shows the nested clusters of data points and the cluster similarities. One can obtain the clusters for each level of cluster numbers by examining a dendrogram.

According to its structure, hierarchical clustering is divided into two. *Agglomerative hierarchical clustering* has the idea of merging the clusters iteratively until obtaining a single cluster. That is, each data point is considered to be a cluster at the beginning. In each iteration, clusters are merged according to a predefined criterion. When all clusters are combined, the overall data set becomes the only cluster and the algorithm stops. Agglomerative clustering has a *bottom-up* approach since it starts with clusters with single data points (or N clusters), and obtains one cluster at the end. The second type of hierarchical clustering is the *divisive hierarchical clustering*. This approach splits the clusters in each iteration. It starts with a single cluster, i.e. the whole data set is the only cluster at the beginning. Then clusters are divided according to some condition until each data point becomes a cluster by themselves. Therefore, divisive clustering is a *top-down approach*.

In both of the algorithms above, dividing or merging operations take place based on cluster similarities. Single linkage and complete linkage are the commonly used methods to measure the cluster similarity. In *single linkage*, the distance between two clusters is determined by the distance between two points that gives the smallest pairwise distance obtained from those clusters. On the other hand, complete linkage defines the distance between two clusters as the greatest pairwise distance between two points, one from each cluster.

One of the advantages of hierarchical clustering is that it does not require any parameters. However, it is a computationally costly algorithm since it starts with one cluster and iterates until obtaining N cluster, or vice versa. In addition, when assignment of an object to a cluster is done in an iteration, the cluster that an object belongs to does not change in next iterations since the algorithm does not take into account the possi-

ble assignments to other clusters afterwards. [4] provides the recent developments in hierarchical clustering methods.

2.2 Partitional Clustering

As opposed to hierarchical clustering, partitional clustering defines a center for each cluster and assigns data points to clusters without constructing a hierarchical structure. It is based on optimizing a criterion function. Partitional clustering is an iterative method which stops when the predefined convergence criterion is satisfied.

Clusters are described by centers in partitional clustering. Some of the mainly used center representations are mean and median. When a cluster center is defined as taking the average of the objects in that cluster, then the center representation of it becomes the mean of objects. On the other hand, a cluster center can be obtained by calculating the median of the objects in that cluster.

Partitional clustering is also called *center-based clustering* since the idea is based on cluster centers. In addition to center representation, criterion function to be optimized should be determined. Minimizing the sum of the squared distances between objects and their cluster centers is the most common objective function in partitional clustering.

Partitional methods differ in the assignment type of objects to clusters, and they are divided into two as hard partitioning and soft partitioning.

- **Hard (Crisp) Partitioning**: Each object belongs to only one cluster. Thus, none of the clusters intersects with any other clusters.
- **Soft Partitioning**: Objects are assigned to clusters with a probability. This method allows objects to be assigned to more than one cluster based on their membership values (or probabilities).

Partitional clustering algorithms do not require much memory and their time complexity is low. However, the number of clusters is a parameter that should be specified beforehand. In addition, partitional clustering algorithms start with random cluster

centers so their performance highly depends on the initialization. To overcome this problem, algorithms should be run multiple times.

K-means clustering algorithm introduced in [5] is the most popular hard partitioning clustering approach. The algorithm starts with randomly selected K cluster centers. In each iteration, objects are assigned to a cluster whose center has the shortest distance from them. Then centers are updated, and assignment procedure continues until there is no change in any of the cluster centers. The distance used in K-means algorithm is Euclidean distance (L_2 norm). Its objective is to minimize the sum of squared error, which is equal to minimizing the total squared distance between objects and the clusters that they are assigned to. Cluster center representation that meets the optimality criterion of this objective function is the mean of cluster members. K-means is a simple yet an effective clustering algorithm. However, some of the drawbacks can be listed as its sensitivity to outliers, dependency on initialization, and taking the number of clusters to be constructed as input.

Similar to K-means clustering, K-median algorithm has a similar logic as in K-means algorithm. However, it differs in center representation and objective function it uses. According to K-median clustering, centers are represented as the median of the cluster members. In addition, the algorithm minimizes the sum of the distance between objects and their cluster centers, where the distance metric is Manhattan distance (L_1 norm). Using the distance itself instead of the squared distance makes K-median clustering less sensitive to outliers.

K-medoids clustering is also one of the widely used hard partitioning method. One of the major difference of K-medoids approach is that its cluster center representation. Instead of calculating the centers as mean or median of cluster members, data points become cluster centers. That is, it selects K data points as cluster centers. In addition, sum of the Manhattan distance (L_1 norm) between the objects and cluster centers is its objective to be minimized. Partitioning Around Medoids (PAM) algorithm proposed in [6] is the most common K-medoids clustering method in the literature. Minimizing not the squared distance but the distance itself yields the algorithm to be more robust to outliers.

Soft partitional clustering algorithms assigns objects to clusters with a probability.

Therefore, they are called *soft clustering methods*. From soft clustering methods, Probabilistic Distance Clustering (PDC) and other probabilistic clustering approaches such as Fuzzy c-Means (FCM), Gustafson Kessel Algorithm (GK), and Gath-Geva clustering algorithm (GG) will be mentioned in this section.

2.2.1 Probabilistic D-Clustering

Presented by Ben-Israel and Iyigun in 2008 [2], probabilistic d-clustering (PDC) is a soft partitioning clustering algorithm based on the assumption that the probabilities are inversely proportional to the distance between objects and cluster centers. Here, probability refers to the membership probability of an object to a cluster.

Consider we have N data points and M features, and $\mathbf{x}_i \in \mathbb{R}^M$ represents object $i \in 1, ...N$. Let the number of clusters be T and $\mathbf{c}_j$ be the center of cluster $j \in 1, ..., T$. Then PDC has two principles as follows:

Principle 1. For each object $\mathbf{x}_i$,

$$p_{ij}\, d_j(\mathbf{x}_i) = D(\mathbf{x}_i)\,, \tag{2.2.1}$$

where $d_j(\mathbf{x}_i)$ is the distance between object i and cluster j, which is $\|\mathbf{x}_i - \mathbf{c}_j\|$, p_{ij} is the probability of assigning object i to cluster j, and $D(\mathbf{x}_i)$ is a constant depending only on data point $\mathbf{x}_i$. This principle states that given cluster centers, the closer an object is to a cluster center, the higher probability of being assigned to that cluster.

Principle 2. For each $\mathbf{x}_i$, probabilities add up to 1. That is,

$$\sum_{j=1}^{T} p_{ij} = 1 \qquad\qquad \forall i$$

Based on Principle 1 and 2, the membership probabilities of data point $\mathbf{x}_i$ are found as

$$p_{ij} = \frac{\dfrac{1}{d_j(\mathbf{x}_i)}}{\displaystyle\sum_{t=1}^{T} \frac{1}{d_t(\mathbf{x}_i)}}\,. \tag{2.2.2}$$

The optimization problem of PDC is defined as

$$\begin{aligned} \min \quad & \sum_{i=1}^{N}\sum_{j=1}^{T} d_j(\mathbf{x}_i)\, p_{ij}^2 \\ \text{s.t.} \quad & \sum_{j=1}^{T} p_{ij} = 1 && \forall i \\ & p_{ij} \geq 0 && \forall i, j \end{aligned}$$

Given the centers, the above problem becomes a convex optimization problem, and optimal probabilities can be found using Lagrangian method. Optimality condition of p_{ij}'s matches (2.2.1) in Principle 1.

Given the probabilities, the optimality condition for cluster center $\mathbf{c}_j$ is obtained as

$$\mathbf{c}_j = \frac{\sum_{i=1}^{N} v_j(\mathbf{x}_i)\, \mathbf{x}_i}{\sum_{i=1}^{N} v_j(\mathbf{x}_i)}, \tag{2.2.3}$$

where

$$v_j(\mathbf{x}_i) = \frac{p_{ij}^2}{d_j(\mathbf{x}_i)}.$$

Given the number of clusters as T, PDC algorithm starts with randomly selected T cluster centers. In each iteration, it updates the cluster centers as in (2.2.3) and probabilities as in (2.2.2) until a termination criterion is satisfied.

PDC is a robust algorithm since it uses the distance itself, not the squared distances. Therefore, the algorithm is not sensitive to outliers. In addition, it is fast and easy to apply. Some drawbacks of PDC are that it may not work well on the non-spherical shaped clusters and data sets where the cluster size are not equal.

PDC algorithm takes the cluster sizes as given. Therefore, it works well when the cluster sizes are equal. To overcome the problem of clustering non-equal clusters, Iyıgun and Ben-Israel [7] come up with a modified PDC algorithm, called PDQ algorithm, that takes the cluster sizes into account. Let q_j be the cluster size of cluster

j. Principle 1 in (2.2.1) is arranged such that for each object $\mathbf{x}_i$

$$\frac{p_{ij}\, d_j(\mathbf{x}_i)}{q_k} = D(\mathbf{x}_i)\,.$$

The idea is that cluster membership probability is proportional to cluster size and inversely proportional to the distance between cluster center and object. The proposed algorithm also copes with unknown cluster sizes by estimating them.

Iyigun and Ben-Israel [8] also study semi-supervised probabilistic distance clustering. The proposed approach combines the clustering and classification approaches. It gives weights to both clustering and classification problem. Besides semi-supervised PDC, Caner [9] develops two-mode probabilistic distance clustering and two novel algorithms for soft two-mode clustering problem.

2.2.2 Other Soft Partitioning Methods

Fuzzy c-Means (FCM), Gustafson-Kessel (GK), and Gath-Geva (GG) clustering algorithms are the well known probabilistic clustering approaches that will be explained in this section.

2.2.2.1 Fuzzy c-Means Algorithm

Based on the fuzzy clustering methods proposed by Bezdek in 1973 [10], Fuzzy c-Means clustering algorithm (FCM) was introduced by Bezdek et al. in 1984 [11]. The algorithm allows data points to be assigned more than one clusters. It is based on taking into account the similarity between data points and clusters via a membership function.

The objective function that FCM minimizes is

$$J_{FCM} = \sum_{i=1}^{N}\sum_{j=1}^{T} u_{ij}^{m} \left\|\mathbf{x}_i - \mathbf{c}_j\right\|^2, \tag{2.2.4}$$

where u_{ij} represents the membership value of object i being assigned to cluster j, $\|.\|$ is the Euclidean distance, and m is the weighting exponent of membership function,

where $m \geq 1$. In addition, the membership function u_{ij} should satisfy the following constraints:

$$\sum_{j=1}^{T} u_{ij} = 1 \qquad \forall i \tag{2.2.5}$$

$$0 \leq u_{ij} \leq 1 \qquad \forall i,\, j \tag{2.2.6}$$

Constraint (2.2.5) assures that the membership probability of object i belonging to clusters should add up to 1. Since membership values are the probabilities, constraint (2.2.6) ensures that the probability values are between 0 and 1.

When (2.2.4) is minimized subject to constraints (2.2.5) and (2.2.6), the optimal values for fuzzy memberships are found as

$$u_{ij} = \left(\sum_{t=1}^{T} \left(\frac{d_{ij}}{d_{it}} \right)^{2/(m-1)} \right)^{-1} \qquad \forall i,\, j \tag{2.2.7}$$

where d_{it} refers to the Euclidean distance between object i and cluster t. Optimal cluster centers become

$$\mathbf{c}_j = \frac{\sum_{i=1}^{N} u_{ij}^m \, \mathbf{x}_i}{\sum_{i=1}^{N} u_{ij}^m} \qquad \forall j\,. \tag{2.2.8}$$

The algorithm starts with predefined number of clusters and $N \times T$ fuzzy membership matrix where i^{th} row j^{th} column element of the matrix refers to u_{ij}. In each iteration, it first updates the cluster centers according to (2.2.8), and then new fuzzy partitions are calculated using (2.2.7) until the termination criterion is satisfied.

One of the disadvantages of the algorithm is that it is computationally expensive. In fact, it is sensitive to noise and suffers from the initialization since it is highly dependent on the initial fuzzy partition matrix.

2.2.2.2 Gustafson-Kessel Algorithm

Gustafson-Kessel algorithm (GK), proposed by Gustafson and Kessel in 1979 [12] is an adapted version of FCM, where Mahalanobis distance is used as a metric. The

main advantage of using Mahalanobis distance over Euclidean distance is that the former creates ellipsoidal shaped clusters whereas the latter forms spherical shapes of clusters. Then optimization problem GK proposes is

$$\min \qquad J_{GK} = \sum_{i=1}^{N} \sum_{j=1}^{T} u_{ij}^{m} \left\| \mathbf{x}_i - \mathbf{c}_j \right\|_M^2 \tag{2.2.9}$$

$$\text{s.t.} \qquad \sum_{j=1}^{T} u_{ij} = 1 \qquad \forall i$$

$$0 \leq u_{ij} \leq 1 \qquad \forall i, j$$

Here u_{ij} is membership value and m is the weighting exponent term as in FCM. The distance term $\|.\|_M^2$ in (2.2.9) refers to the squared Mahalanobis distance defined as

$$\left\| \mathbf{x}_i - \mathbf{c}_j \right\|_M^2 = \left(\mathbf{x}_i - \mathbf{c}_j \right)^T \mathbf{A}_j \left(\mathbf{x}_i - \mathbf{c}_j \right),$$

where $\mathbf{A}_j$ is a positive definite matrix. It is calculated using fuzzy covariance matrix $\mathbf{C}_j$ as follows:

$$\mathbf{A}_j = \left(\rho \left| \mathbf{C}_j \right| \right)^{1/N} \mathbf{C}_j^{-1},$$

where $\rho > 0$ is a scaling parameter. Fuzzy covariance matrix for cluster j is equal to

$$\mathbf{C}_j = \frac{\sum\limits_{i=1}^{N} u_{ij}^{m} (\mathbf{x}_i - \mathbf{c}_j)(\mathbf{x}_i - \mathbf{c}_j)^T}{\sum\limits_{i=1}^{N} u_{ij}^{m}}. \tag{2.2.10}$$

Minimizing (2.2.9) with respect to $\mathbf{c}_j$ yields

$$\mathbf{c}_j = \frac{\sum\limits_{i=1}^{N} u_{ij}^{m} \mathbf{x}_i}{\sum\limits_{i=1}^{N} u_{ij}^{m}}. \tag{2.2.11}$$

Given cluster centers, optimal membership value u_{ij} that minimizes (2.2.9) is obtained as

$$u_{ij} = \left(\sum_{t=1}^{T} \left(\frac{\left(\mathbf{x}_i - \mathbf{c}_j \right)^T \mathbf{A}_j \left(\mathbf{x}_i - \mathbf{c}_j \right)}{\left(\mathbf{x}_i - \mathbf{c}_t \right)^T \mathbf{A}_t \left(\mathbf{x}_i - \mathbf{c}_t \right)} \right)^{1/(m-1)} \right)^{-1}.$$

2.2.2.3 Gath-Geva Clustering Algorithm

Gath-Geva clustering algorithm (GG) is an adaptation of FCM proposed by Gath and Geva in 1989 [13]. It both uses the ideas in FCM and fuzzy maximum likelihood estimation. GG defines the optimization problem as

$$\min \qquad J_{GG} = \sum_{i=1}^{N} \sum_{j=1}^{T} u_{ij}^{m} \, d_{ij} \, , \tag{2.2.12}$$

$$\text{s.t.} \qquad \sum_{j=1}^{T} u_{ij} = 1 \qquad\qquad \forall i$$

$$0 \leq u_{ij} \leq 1 \qquad\qquad \forall i, j$$

where d_{ij} is the distance between $\mathbf{x}_i$ and $\mathbf{c}_j$, which is an exponential distance that takes fuzzy covariance matrix into account. The distance formula is given as

$$d_{ij} = \frac{\sqrt{|\mathbf{C}_j|}}{\frac{1}{N} \sum_{i=1}^{N} u_{ij}} \exp\left(\frac{1}{2} (\mathbf{x}_i - \mathbf{c}_j)^T \, \mathbf{C}_j^{-1} \, (\mathbf{x}_i - \mathbf{c}_j) \right)$$

where $\mathbf{C}_j$ is the fuzzy covariance matrix for cluster j as in (2.2.10). Taking the derivative of the Lagrangian of (2.2.12) with respect to u_{ij} and making it equal to zero yields

$$u_{ij} = \left(\sum_{t=1}^{T} \left(\frac{d_{ij}}{d_{it}} \right)^{1/(m-1)} \right)^{-1} ,$$

which gives the optimal fuzzy membership values. The optimal cluster centers are found as in (2.2.11).

2.3 Kernels

A *metric* on a set $\mathbf{X}$ is a function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that satisfies

i. $d(\mathbf{x}, \mathbf{y}) > 0$ and $d(\mathbf{x}, \mathbf{y}) = 0$ only if $\mathbf{x} = \mathbf{y}$.

ii. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$.

iii $d(\mathbf{x}, \mathbf{y}) \leq d(\mathbf{x}, \mathbf{z}) + d(\mathbf{z}, \mathbf{y})$, for all $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathcal{X}$.

A *metric space* is a vector space with a metric. A metric space is called *complete* if every Cauchy sequence converges to a point in that space.

Inner product space is a vector space with inner product operator. Inner product between vectors $\mathbf{x}$ and $\mathbf{y}$, shown by $\langle \mathbf{x}, \mathbf{y} \rangle$ is calculated as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{m} x_i \, y_i \, ,$$

for vector space $\mathbb{R}^m$. Inner product is considered as one of the similarity measures in machine learning.

Hilbert space, denoted by $\mathscr{H}$, is a complete inner product space [14]. Norm of a vector $\mathbf{f}$ in the Hilbert space is defined as

$$\|\mathbf{f}\|_2 = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle} \, .$$

The Euclidean distance between vectors $\mathbf{f}$ and $\mathbf{g}$ in Hilbert space can be calculated as

$$\|\mathbf{f} - \mathbf{g}\| = \sqrt{\langle \mathbf{f}, \mathbf{f} \rangle - 2 \langle \mathbf{f}, \mathbf{g} \rangle + \langle \mathbf{g}, \mathbf{g} \rangle} \, .$$

A matrix which consists of inner products of a set of vectors is called Gram matrix. Let the vector set $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$, $\mathbf{x}_i \in \mathbb{R}^m$ for all $i = 1, ..., n$ be represented by the matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \ldots & \mathbf{x}_n \end{bmatrix} \, .$$

Then the Gram matrix will be

$$G = \begin{bmatrix} \mathbf{x}_1^T \mathbf{x}_1 & \mathbf{x}_1^T \mathbf{x}_2 & \ldots & \mathbf{x}_1^T \mathbf{x}_n \\ \mathbf{x}_2^T \mathbf{x}_1 & \mathbf{x}_2^T \mathbf{x}_2 & \ldots & \mathbf{x}_2^T \mathbf{x}_n \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_n^T \mathbf{x}_1 & \mathbf{x}_n^T \mathbf{x}_2 & \ldots & \mathbf{x}_n^T \mathbf{x}_n \end{bmatrix} \, ,$$

which is equal to $\mathbf{X}^T \mathbf{X}$.

Let the space of the vectors $\mathbf{x}_i$'s be called as *original space*, and ϕ be a mapping function, where

$$\phi : \mathbf{x} \rightarrow \phi(\mathbf{x}), \;\; \phi(\mathbf{x}) \in \mathscr{H} \, . \tag{2.3.1}$$

When matrix $\mathbf{X}$ is mapped from the original space into $\mathscr{H}$ via ϕ function, the matrix of the mapped data points will be

$$\mathbf{X}_\phi = \begin{bmatrix} \phi(\mathbf{x}_1) & \phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_n) \end{bmatrix} .$$

Then Gram matrix of $\mathbf{X}_\phi$ becomes

$$\mathbf{G}_\phi = \begin{bmatrix} \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_1) & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_1)^T\phi(\mathbf{x}_n) \\ \phi(\mathbf{x}_2)^T\phi(\mathbf{x}_1) & \phi(\mathbf{x}_2)^T\phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_2)^T\phi(\mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_1) & \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_2) & \dots & \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_n) \end{bmatrix} .$$

$\mathbf{G}_\phi$ gives the pairwise inner product of $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ in the Hilbert space, while G provides the inner products in the original space.

2.3.1 Kernel Function

Let k be a function that measures the pairwise similarity of the data points such that

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R} ,$$

where it is a symmetric and non-negative function. Then k is called *kernel function* [15].

Let kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be defined as

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T\phi(\mathbf{y}) . \tag{2.3.2}$$

That is, domain of k is the data points in the original space and it returns with the inner product of them in the Hilbert space. Then the elements of Gram matrix in the Hilbert space can be represented by function k, and $\mathbf{G}_\phi$ becomes

$$\mathbf{G}_\phi = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & k(\mathbf{x}_1, \mathbf{x}_2) & \dots & k(\mathbf{x}_1, \mathbf{x}_n) \\ k(\mathbf{x}_2, \mathbf{x}_1) & k(\mathbf{x}_2, \mathbf{x}_2) & \dots & k(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ k(\mathbf{x}_n, \mathbf{x}_1) & k(\mathbf{x}_n, \mathbf{x}_2) & \dots & k(\mathbf{x}_n, \mathbf{x}_n) \end{bmatrix} .$$

If $\mathbf{G}_\phi$ is a *symmetric positive semi-definite matrix*, then function k is said to be a *symmetric positive semi-definite function*. However, not every function k can be represented as in (2.3.2) since there exists some conditions that function k should satisfy.

2.3.2 Mercer Kernels

Mercer condition states that there exists a mapping function as in (2.3.1) such that the inner products in $\mathscr{H}$ can be written in terms of $k(\mathbf{x}, \mathbf{y})$ only if k is a symmetric positive semi-definite function. In other words, a symmetric positive semi-definite function $k(\mathbf{x}, \mathbf{y})$ can be represented as

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^T \phi(\mathbf{y}) \,.$$

A function with domain $\mathcal{X} \times \mathcal{X}$ and image $\mathbb{R}$ is called *Mercer kernel* if it satisfies the Mercer condition.

Mercer kernels allow us to calculate the inner products in the Hilbert space using data points in the original space. For example, let $\mathbf{x}$, $\mathbf{y}$ be in $\mathbb{R}^2$ and ϕ be a mapping function from $\mathbb{R}^2$ to $\mathbb{R}^3$, where

$$\phi : \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}\, x_1\, x_2 \end{bmatrix} . \tag{2.3.3}$$

Then $\phi(\mathbf{x}) = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix}^T$ and $\phi(\mathbf{y}) = \begin{bmatrix} y_1^2 & y_2^2 & \sqrt{2}\, y_1\, y_2 \end{bmatrix}^T$. The inner product of $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ (mapped data points) is

$$\begin{aligned} \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = \phi(\mathbf{x})^T \phi(\mathbf{y}) &= \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix} \begin{bmatrix} y_1^2 \\ y_2^2 \\ \sqrt{2}\, y_1\, y_2 \end{bmatrix} \\ &= x_1^2\, y_1^2 + x_2^2\, y_2^2 + 2\, x_1\, y_1\, x_2\, y_2 \\ &= (x_1\, y_1 + x_2\, y_2)^2 \,. \end{aligned} \tag{2.3.4}$$

Note that $x_1\, y_1 + x_2\, y_2$ is the inner product of $\mathbf{x}$ and $\mathbf{y}$, $\langle \mathbf{x}, \mathbf{y} \rangle$. Then (2.3.4) is equal to $\langle \mathbf{x}, \mathbf{y} \rangle^2$, or $(\mathbf{x}^T\, \mathbf{y})^2$, so if $k(\mathbf{x}, \mathbf{y})$ is defined as $(\mathbf{x}^T\, \mathbf{y})^2$ it will give the inner product of $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$, i.e. $\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = k(\mathbf{x}, \mathbf{y})$. That is, using a function that takes an inner product in the original space as input, one can obtain an inner product of mapped data points via ϕ function. This is called *kernel trick* in the literature. It allows us to obtain the inner products without explicitly mapping the data into a new space. Note that the idea of mapping the points via ϕ function is to project the data set into a higher dimensional space and to separate the data linearly in the mapped

space. On the other side, since mapping into a higher dimensional space increases the computational complexity it would be costly to calculate the inner products in the mapped space. However, the inner products in the mapped space can be obtained by using the inner products in the original space with the use of kernel trick. Therefore, it reduces the computational complexity.

Using kernel trick, the squared Euclidean distance between $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ is calculated as

$$\begin{aligned}\|\phi(\mathbf{x}) - \phi(\mathbf{y})\|^2 &= \langle \phi(\mathbf{x}) - \phi(\mathbf{y}), \phi(\mathbf{x}) - \phi(\mathbf{y}) \rangle \\ &= \langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle - 2 \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle + \langle \phi(\mathbf{y}), \phi(\mathbf{y}) \rangle \\ &= k(\mathbf{x}, \mathbf{x}) - 2\, k(\mathbf{x}, \mathbf{y}) + k(\mathbf{y}, \mathbf{y})\,.\end{aligned}$$

Here note that the Euclidean distance in the mapped space is calculated with function k without explicitly knowing ϕ function.

Some examples of commonly used kernels are listed below.

a. **Polynomial kernel**

   Polynomial kernel is represented by

   $$k(\mathbf{x}, \mathbf{y}) = (a + \mathbf{x}^T \mathbf{y})^b\,, \tag{2.3.5}$$

   where $a \geq 0$, $b \in \mathbb{N}$. Polynomial kernel is a Mercer kernel function. When a is 0, (2.3.5) is called *homogeneous polynomial kernel*. If a takes a value that is greater than 0, (2.3.5) becomes *inhomogeneous polynomial kernel*.

b. **Gaussian Kernel**

   Gaussian kernel is defined as

   $$k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\,\sigma^2}\right)\,,$$

   where $\sigma > 0$, which is one of the most commonly used kernel function. Gaussian kernel is a Mercer kernel function. It is an example of radial basis function (RBF) kernels. RBF kernels are functions of the distance between data points in the original space, i.e. they are of the form

   $$k(\mathbf{x}, \mathbf{y}) = f(d(\mathbf{x}, \mathbf{y})),$$

where d is a distance metric in the original space and f is a function on $\mathbb{R}$. The inner product of a point with itself is equal to 1 when Gaussian kernel is used, i.e. $k(\mathbf{x}, \mathbf{x}) = 1$. When the Gaussian kernel is used, data points are mapped into an infinite dimensional space.

c. **Sigmoid (Hyperbolic Tangent) Kernel**

The general formula for sigmoid kernel is

$$k(\mathbf{x}, \mathbf{y}) = \tanh(\alpha\, \mathbf{x}^T \mathbf{y} + \nu)\,,$$

where $\alpha > 0$ and $\nu \leq 0$. Sigmoid kernel is not a positive semi-definite function [16]. Therefore, it is not a Mercer kernel. However, it is widely used in practice and gives good results. The distance between two data points sometimes can be less than 0 when sigmoid kernel is used depending on the data points and parameter values. For instance, let $\mathbf{x} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\mathbf{y} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$, $\alpha = 1$, and $\nu = -1$. Then the distance between $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$ becomes

$$\begin{aligned} \|\phi(\mathbf{x}) - \phi(\mathbf{y})\| &= \sqrt{k(\mathbf{x}, \mathbf{x}) - 2\, k(\mathbf{x}, \mathbf{y}) + k(\mathbf{y}, \mathbf{y})} \\ &= \sqrt{\tanh(\mathbf{x}^T \mathbf{x}) - 2\, \tanh(\mathbf{x}^T \mathbf{y}) + \tanh(\mathbf{y}^T \mathbf{y})} \\ &\approx \sqrt{0 - 1.9987 + 1}\,, \end{aligned}$$

which does not satisfy the metric condition which states that the distance should be a non-negative real number.

Data points can be mapped into a higher dimensional space using kernel functions. When the data set is nonlinearly separable, it is difficult to separate the data using linear methods. The aim of kernel methods is to separate the data set in a higher dimensional space where the data becomes linearly separable. Thus, in the new dimension, the data can be separated linearly. This motivates that there may exist a higher dimensional space for a data set where it can be separated linearly. For instance, consider the data set in Figure 2.1. There are 20 data points where half of them is labeled 0 and the other half has label 1. When the data set needs to be divided into two based on their labels, it cannot be done linearly since the data set is non-linearly separable.

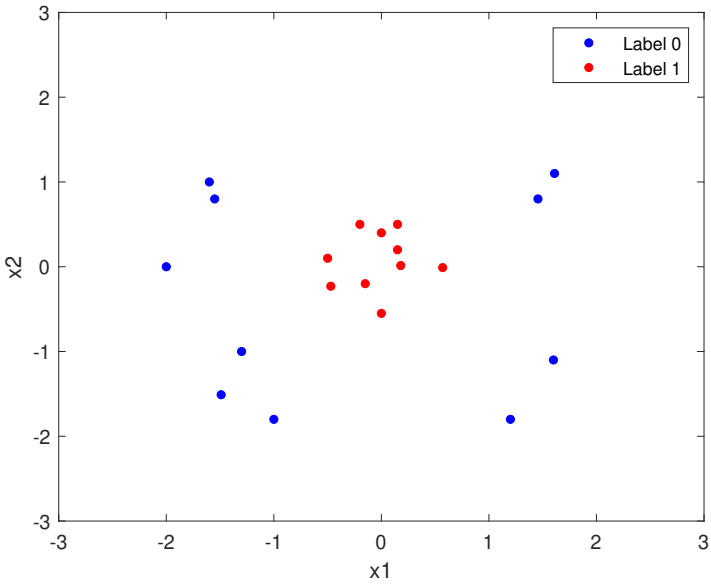


Figure 2.1: Example for a data not linearly separable

We can map the data set using a ϕ function from two-dimension to three-dimension as in (2.3.3). That is,

$$\phi : \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T \rightarrow \begin{bmatrix} z_1 & z_2 & z_3 \end{bmatrix}^T = \begin{bmatrix} x_1^2 & x_2^2 & \sqrt{2}\, x_1\, x_2 \end{bmatrix}^T$$

Mapped data points in the new space is shown in Figure 2.2. Note that now the data can be linearly separable in this mapped space.

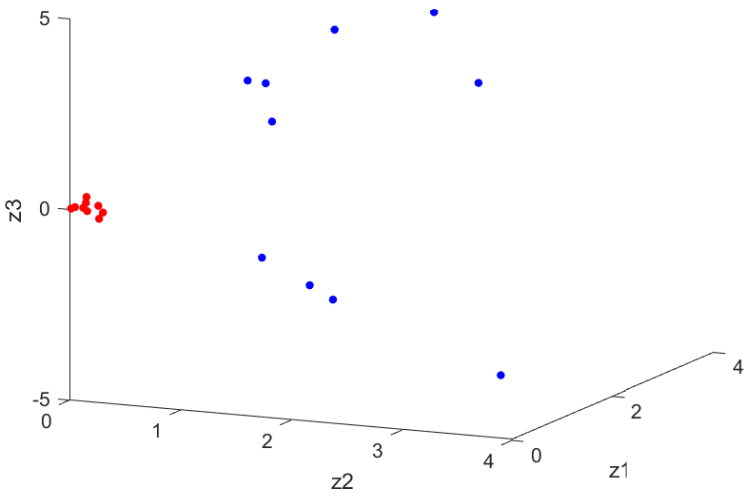


Figure 2.2: Mapped Data

Input space, or original space, implies the space that the data set is defined at the beginning. Kernel space, mapped space, and feature space refer to the higher dimensional space that data is mapped into, and will be used interchangeably in this study.

CHAPTER 3

LITERATURE REVIEW ON KERNEL CLUSTERING METHODS

Kernel functions in machine learning is started to be used in [17] in 1964. Cortes and Vapnik proposed support vector machines (SVM) in 1995, a classification algorithm where the data is mapped into a higher dimensional space using a nonlinear mapping function [18]. In this higher dimensional space, the data was able to be linearly separable. SVM performs better than other classification algorithms. This success broadened the usage of kernel methods in other algorithms.

Kernel k-means is an adaptation of kernel methods to k-means algorithm [19, 20]. The basic idea of kernel k-means algorithm is to apply k-means clustering in the mapped space. The algorithm aims at minimizing the sum of squared error in the kernel space. Given the data set $\mathbf{X}$, mapping function ϕ, and number of clusters T, data set is mapped into kernel space via ϕ function and clusters are constructed by randomly assigning data points. Cluster center $\mathbf{c}_j^\phi$ is calculated by taking the average of data points belonging to that cluster as follows

$$\mathbf{c}_j^\phi = \frac{1}{|A_j|} \sum_{i \in A_j} \phi(\mathbf{x}_i) \qquad \forall j \qquad (3.0.1)$$

where A_j is the set of data instances that is assigned to cluster j. Then based on the squared Euclidean distance between data points and cluster centers in the kernel space, each data point is assigned to the closest cluster. Considering the assignment procedure, cluster centers are recalculated as in (3.0.1). The process continues until there is no change in the cluster centers. The major challenge in this method is to measure the distance between cluster centers and data points. Since kernel methods assume that ϕ function is not known explicitly, data points in the kernel space and cluster centers in (3.0.1) cannot be known. Therefore, the distance cannot be calcu-

lated. However, kernel trick overcomes this problem. The distance between $\phi(\mathbf{x}_i)$ and $\mathbf{c}_j^{\phi}$ is obtained as

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j^{\phi}\|^2 = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\,\phi(\mathbf{x}_i)^T \mathbf{c}_j^{\phi} + (\mathbf{c}_j^{\phi})^T \mathbf{c}_j^{\phi}\,. \tag{3.0.2}$$

Substituting (3.0.1) into (3.0.2) gives

$$\begin{aligned}\|\phi(\mathbf{x}_i) - \mathbf{c}_j^{\phi}\|^2 = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\,\phi(\mathbf{x}_i)^T \frac{1}{|A_j|}\sum_{k\in A_j}\phi(\mathbf{x}_k) \\ + \left(\frac{1}{|A_j|}\sum_{k\in A_j}\phi(\mathbf{x}_k)^T\right)\left(\frac{1}{|A_j|}\sum_{k\in A_j}\phi(\mathbf{x}_k)\right)\,.\end{aligned} \tag{3.0.3}$$

Using kernel trick and rearranging (3.0.3), the distance is found as

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j^{\phi}\|^2 = k(\mathbf{x}_i,\mathbf{x}_i) - 2\,\frac{1}{|A_j|}\sum_{k\in A_j} k(\mathbf{x}_i,\mathbf{x}_k) + \frac{1}{|A_j|^2}\sum_{k\in A_j}\sum_{h\in A_j} k(\mathbf{x}_k,\mathbf{x}_h)\,,$$

where $k(\mathbf{x}_i,\mathbf{x}_k) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_k)$. That is, without explicitly knowing the cluster centers in the kernel space, the distance between data points and cluster centers can still be calculated.

FCM algorithms based on kernel methods can be divided into two categories based on the space that the cluster centers are defined, i.e. whether they are created in the input space or in the kernel space initially. When the cluster centers are constructed in the kernel space, they can only be known implicitly since we cannot obtain any data point in the kernel space explicitly. Therefore, when the distance between a cluster center and a data point is calculated, it will be enough to map the data point into kernel space since centers are already been defined on the kernel space. In this study, fuzzy kernel algorithms that use this cluster definition will be called KFCM-K algorithms. On the other hand, cluster centers can be created in the input space initially. In this case, both the data points and the cluster centers belong to the input space. Then the distance function will be kernelized since the proximity between two objects in the kernel space will be measured by mapping them from the input space to kernel space. We refer to the algorithms that use this approach as KFCM-F in general.

3.1 Literature of KFCM-F based Algorithms

KFCM-F algorithm is the adaptation of FCM algorithm in the kernel space where cluster centers are constructed in the original space. Then centers are mapped into the kernel space via kernel function. The objective to be minimized is given as

$$\sum_{i=1}^{N}\sum_{j=1}^{T} u_{ij}^{m}\|\phi(\mathbf{x}_i)-\phi(\mathbf{c}_j)\|^2,$$

which can be rewritten as

$$\sum_{i=1}^{N}\sum_{j=1}^{T} u_{ij}^{m}\left(k(\mathbf{x}_i,\mathbf{x}_i)-2\,k(\mathbf{x}_i,\mathbf{c}_j)+k(\mathbf{c}_j,\mathbf{c}_j)\right) \tag{3.1.1}$$

using kernel trick. This objective minimizes the sum of squared error in the kernel space. Note that $\mathbf{c}_j$ is the cluster center in the original space whereas $\phi(\mathbf{c}_j)$ refers to the mapped cluster center. The objective is just the kernelization of the metric used in FCM. This method updates the cluster centers in the original space, not the image of them in the kernel space. Therefore, in order to find the cluster centers that minimize (3.1.1), kernel function to be used should be known beforehand since $\mathbf{c}_j$ in the objective function depends on function k.

KFCM-F algorithm works as follows. Cluster centers are initially created in the input space. Then they are mapped into kernel space. According to distance between mapped cluster centers and mapped data points, fuzzy partition matrix is updated and cluster centers in the input space are recalculated. The algorithm goes on until termination criterion is satisfied. Using this method, final cluster centers in the input space are obtained. Shortly, KFCM-F algorithm defines and updates the cluster centers in the input space. Using the image of those centers in the kernel space, it updates the fuzzy matrix.

Zhang and Chen [21] propose KFCM-F algoritm and kernel possibilistic c-means algorithm (KPCM). They use Gaussian kernel function in their algorithms and give the update rules based on this function. They compare the robustness of kernel-based algorithms with FCM and PCM using a data set with outliers. They reported that kernel-based methods yield better results.

Wu et al. [22] develop KFCM-F algorithm where the cluster centers are updated im-

plicitly. They still define cluster centers in the input space. However, center equation in the kernel space is embedded into the distance calculated in the kernel space. That is, cluster centers in the kernel space are defined as

$$\phi(\mathbf{c}_j) = \frac{\sum_{i=1}^{N} u_{ij}^m \, \phi(\mathbf{x}_i)}{\sum_{i=1}^{N} u_{ij}^m}.$$

The distance between a cluster center and an object is obtained by substituting the center equation into the distance function as

$$\|\phi(\mathbf{x}_i) - \phi(\mathbf{c}_j)\|^2 = k(\mathbf{x}_i, \mathbf{x}_i) - 2 \sum_{k=1}^{N} \frac{k(\mathbf{x}_i, \mathbf{x}_k) \, u_{kj}^m}{\sum_{t=1}^{N} u_{tj}^m} + \sum_{k=1}^{N} \sum_{l=1}^{N} \frac{u_{kj}^m \, u_{lj}^m \, k(\mathbf{x}_k, \mathbf{x}_l)}{\left(\sum_{t=1}^{N} u_{tj}^m \right)^2}.$$

They experimented the proposed algorithm on ring and spherical data set using second order polynomial kernels, and observed that KFCM-F works well on those data sets.

Zhang and Chen [23] develop a kernel-based FCM and apply their algorithm on MRI data set. They also propose a KFCM-F method with a spatial penalty constraint on fuzzy membership values. This penalty is added to the objective function, and it acts like a regularization term, which aims at dealing with noises in images. Their proposed method is developed for Gaussian kernel. Experiment results show that their algorithms work well on the image data compared to the FCM and FCM with spatial constraints.

Shen et al. [24] propose attribute weighted KFCM-F algorithm. The idea behind this method is that some of the attributes in higher dimension may be irrelevant. Therefore, to reduce the effect of these features on the clustering, they come up with a weighted kernel-based fuzzy clustering algorithm. They also extend their algorithm to be effective on clustering incomplete data. The computational studies support the success of weighted KFCM-F algorithm over FCM and hard clustering methods.

Ding and Fu [25] apply genetic algorithm to KFCM-F algorithm. They determine the initial cluster centers in the input space using genetic algorithm. These centers become the input of the KFCM-F algorithm. With this way, they aim at improving the clustering performance by providing a good initial centers.

3.2 Literature of KFCM-K based Algorithms

Kernel fuzzy clustering algorithms based on kernel space (KFCM-K) construct the cluster centers in the mapped space. Therefore, there is no explicit representation of the centers in the input space. In fact, the centers in the kernel space are also not known. Basically, these algorithms use the following objective function

$$\sum_{i=1}^{N}\sum_{j=1}^{T} u_{ij}^{m}\|\phi(\mathbf{x}_i)-\mathbf{c}_j^{\phi}\|^2\,,$$

where $\mathbf{c}_j^{\phi}$ refers to the cluster center which is defined in the kernel space. Note that only the data points are mapped since clusters already belong to kernel space. The distance term in the objective cannot be rewritten using kernel trick since kernel trick can only be used for the case where the data points in the original space are known explicitly, and we do not know the inverse image of cluster centers in the input space.

Optimal fuzzy membership values are obtained as

$$u_{ij}=\left(\sum_{t=1}^{T}\left(\frac{\|\phi(\mathbf{x}_i)-\mathbf{c}_j^{\phi}\|^2}{\|\phi(\mathbf{x}_i)-\mathbf{c}_t^{\phi}\|^2}\right)^{1/(m-1)}\right)^{-1} \qquad \forall i,\ j \qquad (3.2.1)$$

Given the cluster centers, optimal cluster centers are found as

$$\mathbf{c}_j^{\phi}=\frac{\sum_{i=1}^{N} u_{ij}^{m}\,\phi(\mathbf{x}_i)}{\sum_{i=1}^{N} u_{ij}^{m}}\cdot$$

Note that the center equation above cannot be obtained since $\phi(\mathbf{x}_i)$ are not known. Therefore, KFCM-K methods embed the center equation into the distance terms in (3.2.1), where the distance becomes

$$\|\phi(\mathbf{x}_i)-\mathbf{c}_t^{\phi}\|^2=k(\mathbf{x}_i,\mathbf{x}_i)-2\sum_{k=1}^{N} u_{kj}^{m}\,k(\mathbf{x}_i,\mathbf{x}_k)+\sum_{k=1}^{N}\sum_{l=1}^{N} u_{kj}^{m}\,u_{lj}^{m}\,k(\mathbf{x}_k,\mathbf{x}_l)\cdot \qquad (3.2.2)$$

With this way, fuzzy partition values are calculated without obtaining the cluster centers explicitly.

KFCM-K algorithm starts with random fuzzy partition matrix. Then membership values are updated using (3.2.1) and distances calculated in (3.2.2). Center updates

are made implicitly since the distance term does not contain the centers explicitly. Algorithm continues until fuzzy partition values does not change.

Li et al. [26] obtain KFCM-K algorithm by modifying FCM algorithm. They applied both KFCM-K and FCM on a nonlinearly separable two clustered data set and reported that KFCM-K outperforms FCM.

Zhou and Gan [27] develop KFMC-K algorithm and came up with approximated cluster centers in the input space. The idea is that approximate cluster centers in the input space can be obtained such that the difference between centers in the kernel space and the image of input space centers are minimized. This minimization requires prior knowledge of the kernel function to be used. The authors propose approximate cluster centers for both polynomial and Gaussian kernels.

Zhang and Chen [28] propose KFCM-K algorithm where cluster centers in the kernel space are represented by a combination of all data points. That is, each data point has a coefficient for each cluster center. The algorithm determines the center coefficients and fuzzy membership values iteratively. They also decide on the number of clusters in data sets by examining the behavior of the block-diagonal structure on the kernel matrix.

3.3 Literature of Other Kernel-based FCM Algorithms

Graves and Pedrycz [29] compare kernel-based fuzzy clustering methods with FCM and GK algorithms. From the kernel-based clustering algorithms they use KFCM-K algorithms with both polynomial and Gaussian kernel and KFCM-F algorithm with Gaussian kernel only. They measure the performance on traditional clustering algorithms and kernel fuzzy algorithms on UCI machine learning data sets and synthetic data sets. They conclude that the results of the kernel-based algorithms are similar to that of FCM and GK. However, kernel function outperforms the traditional algorithms on non-spherical shaped data sets.

Filippone et al. [30] provide a literature survey on kernel and spectral clustering methods. They present the kernel version of the already-proposed clustering algo-

rithms. They also focus on the kernel-based fuzzy clustering algorithms. Kernel k-means, kernel fuzzy clustering methods, support vector clustering, kernel self organizing maps are some of the kernel methods that are mentioned in this survey.

One of the challenges of kernel based clustering is the decision of the kernel function to be used. Sometimes the data sets can come from different distributions so that each of them can be represented better with different kernel functions or kernel parameters. To overcome the problem of difficulty in choosing the appropriate kernel functions, multiple kernel algorithms have been proposed.

Baili and Frigui [31] propose multiple kernel KFCM-F algorithm. Their multiple kernels contain Gaussian kernels with different parameters. Each kernel function has a weight, and convex combination of the Gaussian kernels construct multiple kernel function. They experimented the proposed algorithms on data sets containing clusters with different densities and observed that multiple kernel KFCM-F outperforms KFCM-F. Dagher [32] develops multiple kernel KFCM-F algorithm with Gaussian kernels, and aims at optimizing the Gaussian kernel parameters. The method uses an approach that is similar to expectation maximization. Huang et al. [33] come up with a multiple kernel KFCM-K clustering algorithm where kernel functions are not limited to Gaussian kernel only. They also determine the weight of each kernel function.

Zeng et al. [34] provide KFCM-K with Mahalanobis distance. The major difficulty of using Mahalanobis distance in the kernel space is to obtain the inverse of the covariance matrix of features in the kernel space. They subtract logarithmic determinant of the covariance matrix to their objective function as a regularization term. With this way, they obtain the approximate covariance matrix of features in the kernel space.

CHAPTER 4

KERNEL PROBABILISTIC DISTANCE CLUSTERING

4.1 Introduction

This chapter firstly introduces PDC in *kernel space* and its principles. Following that, two kernel-based PDC algorithms will be provided, namely KPD and KPD-F.

Consider we have an $N \times M$ data set $\mathbf{X}$, where N and M represent number of data points and features, respectively. We show data point $\mathbf{x}_i$ as M-dimensional vector, where $i = 1, ..., N$. There are T clusters, and cluster centers (prototypes) are shown by $\mathbf{c}_j$, where $j = 1, ..., T$. Then p_{ij} is the probability of data point $\mathbf{x}_i$ belonging to cluster $\mathbf{c}_j$. Please note that cluster center j in the *kernel space* will be represented by $\mathbf{c}_j$ from now on.

4.2 Probabilistic Distance Clustering in Kernel Space

Given the data set $\mathbf{X}$ and the number of clusters T, $\mathbf{x}_i \in \mathbb{R}^M$ is mapped into a higher dimensional space, *kernel space*, via ϕ function. $\phi(\mathbf{x}_i)$ represents the image of $\mathbf{x}_i$ in the *kernel space*. Centers in the *kernel space* are shown by $\mathbf{c}_j$. Figure 4.1 provides the mapping from *input space* to *kernel space*.

Following the ideas in [2], probabilistic distance clustering starts with two principles. First one states that for given cluster centers, the product of the probability of a data point $\phi(\mathbf{x}_i)$ being a member of cluster $\mathbf{c}_j$, p_{ij}, and the distance between $\phi(\mathbf{x}_i)$ and $\mathbf{c}_j$, $\|\big(\phi(\mathbf{x}_i) - \mathbf{c}_j\big)\|$ or simply $d_j\big(\phi(\mathbf{x}_i)\big)$, is a constant which only depends on $\phi(\mathbf{x}_i)$. We accept that these principles hold in *kernel space*, and we obtain the following:

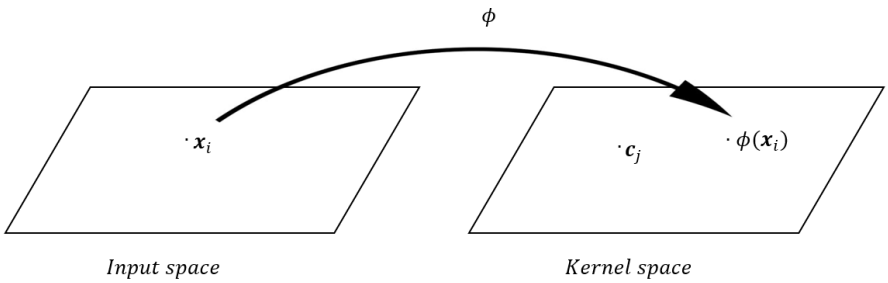


Figure 4.1: Kernel Pd-clustering in Input and Kernel Space

Principle 1. For each data point in the *kernel space*, $\phi(\mathbf{x}_i),\ i = 1, ..., N$,

$$d_j\big(\phi(\mathbf{x}_i)\big)\, p_{ij} = S_i\,, \tag{4.2.1}$$

where S_i is the constant for $\phi(\mathbf{x}_i)$.

Principle 2. Probabilities add up to 1 for each $\phi(\mathbf{x}_i),\ i = 1, ..., N$. That is,

$$\sum_{j=1}^{T} p_{ij} = 1 \quad \forall i\,. \tag{4.2.2}$$

4.2.1 Probabilities

For simplicity, assume there are two clusters, i.e. $T = 2$. Using the principles mentioned above, we can find the membership probabilities of each cluster for each data point as

$$p_{i1} = \frac{d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)},\ p_{i2} = \frac{d_1\big(\phi(\mathbf{x}_i)\big)}{d_2\big(\phi(\mathbf{x}_i)\big) + d_1\big(\phi(\mathbf{x}_i)\big)} \quad \forall i\,. \tag{4.2.3}$$

Proof. We can write

$$p_{i2} = \frac{p_{i1}\, d_1\big(\phi(\mathbf{x}_i)\big)}{d_2\big(\phi(\mathbf{x}_i)\big)} \tag{4.2.4}$$

using (4.2.1). In fact, $p_{i1} + p_{i2} = 1$. Substituting (4.2.4) gives

$$p_{i1} = \frac{d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)}\,. \tag{4.2.5}$$

□

In general, when there are T clusters, the probability of i^{th} data point being a member of cluster j becomes

$$p_{ij} = \frac{\prod\limits_{q \neq j} d_q\big(\phi(\mathbf{x}_i)\big)}{\sum\limits_{t=1}^{T} \prod\limits_{q \neq t} d_q\big(\phi(\mathbf{x}_i)\big)}.$$

4.2.2 The Joint Distance Function

The probabilities in (4.2.1) can be written as

$$p_{ij} = \frac{S_i}{d_j\big(\phi(\mathbf{x}_i)\big)}. \tag{4.2.6}$$

Substituting (4.2.6) into (4.2.2), we get

$$S_i = \frac{d_1\big(\phi(\mathbf{x}_i)\big)\, d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)}. \tag{4.2.7}$$

S_i in (4.2.7) is the joint distance function of $\mathbf{x}_i$, and it is a measure of distance between $\phi(\mathbf{x}_i)$ and all cluster centers. In the case of T clusters, the above equation becomes

$$S_i = \frac{\prod\limits_{q=1}^{T} d_q\big(\phi(\mathbf{x}_i)\big)}{\sum\limits_{t=1}^{T} \prod\limits_{q \neq t} d_q\big(\phi(\mathbf{x}_i)\big)}.$$

4.2.3 An Extremal Principle

Given the cluster centers $\mathbf{c}_1$ and $\mathbf{c}_2$, the probabilities in (4.2.3) are the optimal solution of the minimization problem below.

$$\min \quad \sum_{i=1}^{N} d_1\big(\phi(\mathbf{x}_i)\big)\, p_{i1}^2 + d_2\big(\phi(\mathbf{x}_i)\big)\, p_{i2}^2 \tag{4.2.8}$$

$$\text{s.t.} \quad p_{i1} + p_{i2} = 1 \qquad \forall i$$

$$p_{i1},\, p_{i2} \geq 0 \qquad \forall i$$

Proof. The Lagrangian of the problem is

$$L(\mathbf{P}, \mathbf{\Lambda}) = \sum_{i=1}^{N} d_1\big(\phi(\mathbf{x}_i)\big)\, p_{i1}^2 + d_2\big(\phi(\mathbf{x}_i)\big)\, p_{i2}^2 - \sum_{i=1}^{N} \lambda_i \left(p_{i1} + p_{i2} - 1\right) . \tag{4.2.9}$$

Taking the partial derivative of (4.2.9) with respect to p_{ij} yields

$$\frac{\partial L}{\partial p_{ij}} = 2\, d_j\big(\phi(\mathbf{x}_i)\big)\, p_{ij} - \lambda_i\, p_{ij} . \tag{4.2.10}$$

Making (4.2.10) equal to 0 gives

$$p_{ij} = \frac{\lambda_i}{2\, d_j\big(\phi(\mathbf{x}_i)\big)} . \tag{4.2.11}$$

We know that $p_{i1} + p_{i2} = 1$ and substituting (4.2.11) for the probabilities yields

$$\frac{\lambda_i}{2\, d_1\big(\phi(\mathbf{x}_i)\big)} + \frac{\lambda_i}{2\, d_2\big(\phi(\mathbf{x}_i)\big)} = 1 .$$

Then λ_i is found as

$$\lambda_i = 2\, \frac{d_1\big(\phi(\mathbf{x}_i)\big)\, d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)} . \tag{4.2.12}$$

Using (4.2.12), (4.2.11) for p_{i1} becomes

$$p_{i1} = \frac{2\, \dfrac{d_1\big(\phi(\mathbf{x}_i)\big)\, d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)}}{2\, d_1\big(\phi(\mathbf{x}_i)\big)} = \frac{d_2\big(\phi(\mathbf{x}_i)\big)}{d_1\big(\phi(\mathbf{x}_i)\big) + d_2\big(\phi(\mathbf{x}_i)\big)} ,$$

which gives the same equation as in (4.2.5). □

When there are T clusters, the general version of the above minimization problem becomes

$$\min \qquad \sum_{i=1}^{N} \sum_{j=1}^{T} d_j\big(\phi(\mathbf{x}_i)\big)\, p_{ij}^2 \tag{4.2.13}$$

$$\text{s.t.} \qquad \sum_{j=1}^{T} p_{ij} = 1 \qquad\qquad \forall i$$

$$p_{ij} \geq 0 \qquad\qquad \forall i, j$$

4.2.4 Centers

When Euclidean distance is used, distance function in (4.2.8) becomes

$$d_j\big(\phi(\mathbf{x}_i)\big) = \|\phi(\mathbf{x}_i) - \mathbf{c}_j\|\,, \qquad j = 1, 2\,.$$

For Euclidean distance, the objective function in (4.2.8) can be written as

$$f(\mathbf{c}_1, \mathbf{c}_2) = \sum_{i=1}^{N} \|\phi(\mathbf{x}_i) - \mathbf{c}_1\|\, p_{i1}^2 + \|\phi(\mathbf{x}_i) - \mathbf{c}_2\|\, p_{i2}^2\,. \tag{4.2.14}$$

Given the probabilities and assuming that the centers do not coincide with data points, $\mathbf{c}_1$ and $\mathbf{c}_2$ will be

$$\mathbf{c}_j = \sum_{i=1}^{N} \left(\frac{\dfrac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\displaystyle\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} \right) \phi(\mathbf{x}_i)\,, \qquad j = 1, 2\,. \tag{4.2.15}$$

Proof. The derivative of $\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|$ with respect to $\mathbf{c}_j$ is equal to

$$\nabla_{\mathbf{c}_j} \|\phi(\mathbf{x}_i) - \mathbf{c}_j\| = -\frac{\phi(\mathbf{x}_i) - \mathbf{c}_j}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}, \qquad j = 1, 2$$

when $\phi(\mathbf{x}_i) \neq \mathbf{c}_j \quad \forall i \in \{1, ..., N\}$. Then given the probabilities, the derivative of (4.2.14) with respect to $\mathbf{c}_j$ will be

$$\nabla_{\mathbf{c}_j^k} f(\mathbf{c}_1, \mathbf{c}_2) = -\sum_{i=1}^{N} \frac{\phi(\mathbf{x}_i) - \mathbf{c}_j}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} p_{ij}^2 \qquad j = 1, 2\,. \tag{4.2.16}$$

Making (4.2.16) equal to 0 gives

$$\mathbf{c}_j = \sum_{i=1}^{N} \left(\frac{v_j\big(\phi(\mathbf{x}_i)\big)}{\displaystyle\sum_{t=1}^{N} v_j\big(\phi(\mathbf{x}_t)\big)} \right) \phi(\mathbf{x}_i)\,, \qquad j = 1, 2\,, \tag{4.2.17}$$

where

$$v_j\big(\phi(\mathbf{x}_i)\big) = \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} \qquad j = 1, 2\,.$$

□

Please note that when there are T clusters, (4.2.15) is still valid and can be applied for $j = 1, ..., T$.

We have two groups of decision variables, p_{ij}'s and $\mathbf{c}_j$'s. When centers are known, we solve the problem in (4.2.13). We find the centers using (4.2.17).

4.3 Kernel Probabilistic Distance Clustering Algorithms

The probabilistic distance clustering model in *kernel space* has been introduced in Section 4.2. The generalized version of the model will be provided in this section.

In Kernel Probabilistic Distance Clustering there are two spaces, which are *input space* and *kernel space* (or mapped space). *Input space* is the original space of the data points. On the other hand, *kernel space* represents the space where data points are mapped into via ϕ function. The dimension of the *input space* is the number of features data points have. However, *kernel space*'s dimension is determined by the ϕ function to be used. Since we will not know the ϕ function, we will not be able to detect the dimension of the *kernel space*.

4.3.1 Kernel Pd-clustering in Kernel Space

In Kernel Pd-clustering in *kernel space* (**KPD Algorithm**) centers are defined in the *kernel space* (or mapped space). In fact, center update rules are made in the *kernel space*. The data points are in the *input space*, and they are mapped into the *kernel space* via ϕ function.

Since the update rules in the algorithm are valid for all kernel functions, prior knowledge of the kernel function to be used is not required. In addition, cluster prototypes do not have an explicit representation in both *kernel* and *input spaces*.

As a distance metric Euclidean distance is considered. In general, Euclidean distance between $\mathbf{x}_i$ and $\mathbf{c}_j$, $d(\mathbf{x}_i, \mathbf{c}_j)$, is

$$d(\mathbf{x}_i, \mathbf{c}_j) = \sqrt{\sum_{m=1}^{M} (\mathbf{x}_{im} - \mathbf{c}_{jm})^2} = \|\mathbf{x}_i - \mathbf{c}_j\|.$$

If mapping of $\mathbf{x}_i$ into *kernel space* is considered, then $d(\phi(\mathbf{x}_i), \mathbf{c}_j) = \|\phi(\mathbf{x}_i) - \mathbf{c}_j\|$.

The optimization problem of clustering in Section 4.2 becomes

$$\min \quad \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \|\phi(\mathbf{x}_i) - \mathbf{c}_j\| \tag{4.3.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{T} p_{ij} = 1 \qquad \forall i$$

$$p_{ij} \geq 0 \qquad \forall i, j$$

$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|$ is simply denoted by d_{ij}. The nonlinear objective function is convex in terms of p_{ij} and d_{ij}. This is because when p_{ij}'s are given and d_{ij}'s become unknown, the objective becomes convex. Similarly, given d_{ij}'s, the objective function in (4.3.1) will be a function of d_{ij}'s which is a convex function. Considering the fact that constraints are all linear, i.e. they are also convex, the problem becomes a convex optimization problem.

When $\mathbf{c}_j$'s are given, the Lagrangian becomes

$$L(\mathbf{P}, \mathbf{C}, \mathbf{\Lambda}) = \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \|\phi(\mathbf{x}_i) - \mathbf{c}_j\| - \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{T} p_{ij} - 1 \right). \tag{4.3.2}$$

When we take the derivative with respect to p_{ij} and make it equal to zero, we obtain

$$2\, p_{ij} \|\phi(\mathbf{x}_i) - \mathbf{c}_j\| - \lambda_i = 0$$

$$\implies p_{ij} = \frac{\lambda_i}{2\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}. \tag{4.3.3}$$

When we substitute (4.3.3) into (4.3.2), we get

$$\sum_{j=1}^{T} p_{ij} = 1 \implies \sum_{j=1}^{T} \frac{\lambda_i}{2\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} = 1\,.$$

Then

$$\lambda_i = \frac{1}{\sum\limits_{j=1}^{T} \dfrac{1}{2\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}\,. \tag{4.3.4}$$

If (4.3.4) is substituted into (4.3.3), we find out p_{ij} as

$$p_{ij} = \frac{\dfrac{1}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\displaystyle\sum_{t=1}^{T} \frac{1}{\|\phi(\mathbf{x}_i) - \mathbf{c}_t\|}}.$$

Fixing p_{ij} and taking the derivative of (4.3.2) with respect to $\mathbf{c}_j$ gives

$$\mathbf{c}_j = \frac{\displaystyle\sum_{i=1}^{N} \frac{p_{ij}^2\, \phi(\mathbf{x}_i)}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\displaystyle\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}. \tag{4.3.5}$$

The distance between $\phi(\mathbf{x}_i)$ and $\mathbf{c}_j$ can be calculated as

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\| = \sqrt{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_i) - 2\,\phi(\mathbf{x}_i)^T \mathbf{c}_j + \mathbf{c}_j^T \mathbf{c}_j} \tag{4.3.6}$$

Substituting (4.3.5) and using the *kernel trick*, (4.3.6) can be written as

$$\|\phi(\mathbf{x}_i) - \mathbf{c}_j\| = \sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2\,\frac{\displaystyle\sum_{k=1}^{N} \frac{p_{kj}^2\, k(\mathbf{x}_i, \mathbf{x}_k)}{\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}}{\displaystyle\sum_{k=1}^{N} \frac{p_{kj}^2}{\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}} + \frac{\displaystyle\sum_{k=1}^{N}\sum_{l=1}^{N} \frac{p_{kj}^2\, p_{lj}^2\, k(\mathbf{x}_k, \mathbf{x}_l)}{\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|\,\|\phi(\mathbf{x}_l) - \mathbf{c}_j\|}}{\left(\displaystyle\sum_{k=1}^{N} \frac{p_{kj}^2}{\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}\right)^2}}, \tag{4.3.7}$$

where k is the kernel function. However, the cluster centers cannot be calculated using (4.3.5) since $\phi(\mathbf{x}_i)$'s are not known explicitly. Therefore, there is no explicit representation of cluster centers. In fact, since we cannot calculate the cluster centers, the distance given in (4.3.7) cannot be obtained.

On the other side, the centers can be described in terms of the data points that belong to the corresponding clusters. That is, the centers will be the convex combination of $\phi(\mathbf{x}_i)$'s. Then (4.3.5) can be rewritten as

$$\mathbf{c}_j = \sum_{i=1}^{N} \beta_{ij}\, \phi(\mathbf{x}_i), \tag{4.3.8}$$

where

$$\beta_{ij} = \frac{\dfrac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum\limits_{i=1}^{N} \dfrac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} \,. \tag{4.3.9}$$

For a given cluster index j, when we sum (4.3.9) for each data point, we obtain

$$\sum_{i=1}^{N} \beta_{ij} = \sum_{i=1}^{N} \frac{\dfrac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum\limits_{h=1}^{N} \dfrac{p_{hj}^2}{\|\phi(\mathbf{x}_h) - \mathbf{c}_j\|}} = \frac{\sum\limits_{i=1}^{N} \dfrac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum\limits_{h=1}^{N} \dfrac{p_{hj}^2}{\|\phi(\mathbf{x}_h) - \mathbf{c}_j\|}} = 1 \,. \tag{4.3.10}$$

When (4.3.8) is substituted into $\mathbf{c}_j$'s in the objective function, then the problem becomes

$$\min \qquad \sum_{i=1}^{N} \sum_{j=1}^{T} p_{ij}^2 \left\| \phi\left(\mathbf{x}_i\right) - \sum_{i=1}^{N} \beta_{ij}\, \phi(\mathbf{x}_i) \right\| \tag{4.3.11}$$

$$\text{s.t.} \qquad \sum_{j=1}^{T} p_{ij} = 1 \qquad \forall i$$

$$\sum_{i=1}^{N} \beta_{ij} = 1 \qquad \forall j$$

$$p_{ij} \geq 0 \qquad \forall i, j$$

$$\beta_{ij} \geq 0 \qquad \forall i, j$$

The distance $\left\| \phi\left(\mathbf{x}_i\right) - \sum\limits_{i=1}^{N} \beta_{ij} \phi(\mathbf{x}_i) \right\|$ in the objective function is equal to

$$\sqrt{\phi\left(\mathbf{x}_i\right)^T \phi\left(\mathbf{x}_i\right) - 2 \sum_{k=1}^{N} \beta_{kj}\, \phi\left(\mathbf{x}_i\right)^T \phi\left(\mathbf{x}_k\right) + \sum_{k=1}^{N} \sum_{l=1}^{N} \beta_{kj}\, \beta_{lj}\, \phi\left(\mathbf{x}_k\right)^T \phi\left(\mathbf{x}_l\right)} \,. \tag{4.3.12}$$

We replace $\phi\left(\mathbf{x}_i\right)^T \phi\left(\mathbf{x}_k\right)$'s with $k(\mathbf{x}_i, \mathbf{x}_k)$'s, and 4.3.12 becomes

$$\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2 \sum_{k=1}^{N} \beta_{kj}\, k(\mathbf{x}_i, \mathbf{x}_k) + \sum_{k=1}^{N} \sum_{l=1}^{N} \beta_{kj}\, \beta_{lj}\, k(\mathbf{x}_k, \mathbf{x}_l)} \,. \tag{4.3.13}$$

The objective in (4.3.11) can be represented in terms of matrices and vectors. We can define $\boldsymbol{\beta}_j$ as the column vector containing all β_{ij}'s for a given cluster j, which is $\boldsymbol{\beta}_j = \begin{bmatrix} \beta_{1j} & \beta_{2j} & \dots & \beta_{Nj} \end{bmatrix}^T$. Let $\mathbf{K}$ be the kernel matrix (or Gram matrix in Hilbert space as in Section 2.3), $\mathbf{K}_i$ be the i^{th} column of the matrix, and $\mathbf{K}_{ii}$ be the i^{th} row and i^{th} column element. Then the distance in (4.3.13) will be written as

$$\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j}\,.$$

So, the optimization problem in (4.3.11) becomes

$$\begin{aligned}
\min \quad & \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j} \\
\text{s.t.} \quad & \sum_{j=1}^{T} p_{ij} = 1 && \forall i \\
& \mathbf{1}_N^T \boldsymbol{\beta}_j = 1 && \forall j \\
& p_{ij} \geq 0 && \forall i, j \\
& \beta_{ij} \geq 0 && \forall i, j
\end{aligned}$$

where $\mathbf{1}_N$ is a column vector of length N, whose elements are all 1. The Lagrangian of the problem becomes

$$\begin{aligned}
L(\mathbf{P}, \boldsymbol{\beta}, \boldsymbol{\Lambda}, \boldsymbol{\Gamma}) = & \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j} \\
& - \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{T} p_{ij} - 1 \right) - \sum_{j=1}^{T} \gamma_j \left(\mathbf{1}_N^T \boldsymbol{\beta}_j - 1 \right) .
\end{aligned} \tag{4.3.14}$$

Given β_{ij}'s, taking the derivative of (4.3.14) with respect to p_{ij} and making it equal to zero gives

$$p_{ij} = \frac{\dfrac{1}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j}}}{\displaystyle\sum_{t=1}^{T} \frac{1}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_t^T \mathbf{K}_i + \boldsymbol{\beta}_t^T \mathbf{K} \boldsymbol{\beta}_t}}} \,. \tag{4.3.15}$$

Taking the derivative of (4.3.14) with respect to $\boldsymbol{\beta}_j$ yields

$$\frac{\partial L}{\partial \boldsymbol{\beta}_j} = \frac{1}{2}\sum_{i=1}^{N} p_{ij}^2 \frac{-\mathbf{K}_i + \mathbf{K}\,\boldsymbol{\beta}_j}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}} - \gamma_j\,\mathbf{1}_N\,. \tag{4.3.16}$$

Making (4.3.16) equal to $\mathbf{0}$ gives

$$\begin{aligned} &\mathbf{K}\,\boldsymbol{\beta}_j \sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}} \\ &= \sum_{i=1}^{N} \frac{p_{ij}^2\mathbf{K}_i}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}} + \gamma_j\,\mathbf{1}_N\,. \end{aligned} \tag{4.3.17}$$

If kernel matrix $\mathbf{K}$ is invertible, then (4.3.17) can be multiplied by $\mathbf{K}^{-1}$ from the left, and $\boldsymbol{\beta}_j$ is obtained as

$$\boldsymbol{\beta}_j = \frac{\sum_{i=1}^{N} \frac{p_{ij}^2\,\mathbf{K}^{-1}\,\mathbf{K}_i}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}} + \frac{\gamma_j\,\mathbf{K}^{-1}\,\mathbf{1}_N}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}}\,. \tag{4.3.18}$$

Taking the derivative of (4.3.14) with respect to γ_j and making it equal to 0 gives

$$\frac{\partial L}{\partial \gamma_j} = \mathbf{1}_N^T\,\boldsymbol{\beta}_j - 1 = 0\,. \tag{4.3.19}$$

By substituting (4.3.18) into (4.3.19), we obtain

$$\frac{\sum_{i=1}^{N} \frac{p_{ij}^2\,\mathbf{1}_N^T\,\mathbf{K}^{-1}\,\mathbf{K}_i}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}} + \frac{\gamma_j\,\mathbf{1}_N^T\,\mathbf{K}^{-1}\,\mathbf{1}_N^T}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}} = 1\,. \tag{4.3.20}$$

Note that $\mathbf{K}^{-1}\,\mathbf{K}_i$ is equal to $\mathbf{I}_i$, $\forall i = 1, ..., N$, where $\mathbf{I}_i$ is a column vector of length N whose i^{th} element is 1 and the remaining elements are all 0. Therefore, $\mathbf{1}_N^T\,\mathbf{K}^{-1}\,\mathbf{K}_i = 1$, $\forall i = 1, ..., N$, and the first term in (4.3.20) equals to 1, then the equation becomes

$$\frac{\gamma_j\,\mathbf{1}_N^T\,\mathbf{K}^{-1}\,\mathbf{1}_N}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T\,\mathbf{K}_i + \boldsymbol{\beta}_j^T\mathbf{K}\boldsymbol{\beta}_j}}} = 0\,. \tag{4.3.21}$$

The denominator of (4.3.21) is greater than 0. The term $\mathbf{1}_N^T \mathbf{K}^{-1} \mathbf{1}_N$ in the nominator is the sum of all elements in matrix K^{-1}, which is not equal to 0. Therefore, γ_j becomes 0. Substituting $\gamma_j = 0$ into (4.3.18) gives

$$\boldsymbol{\beta}_j = \frac{\sum_{i=1}^{N} \frac{p_{ij}^2 \mathbf{K}^{-1} \mathbf{K}_i}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j}}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\sqrt{\mathbf{K}_{ii} - 2\boldsymbol{\beta}_j^T \mathbf{K}_i + \boldsymbol{\beta}_j^T \mathbf{K} \boldsymbol{\beta}_j}}}. \tag{4.3.22}$$

$\boldsymbol{\beta}_j$ vectors are updated as in (4.3.22). Using (4.3.15) we update p_{ij}'s. In that formula since $\mathbf{c}_j$'s are substituted with $\boldsymbol{\beta}_j$'s and $\phi(\mathbf{x}_i)$'s, there is still no need to know $\mathbf{c}_j$'s explicitly.

4.3.1.1 KPD Algorithm

KPD Algorithm (Kernel Probabilistic Distance Clustering in Kernel Space) can be developed by determining an update rule for $\boldsymbol{\beta}_j$'s and p_{ij}'s. Since the centers are not known explicitly, they are updated implicitly in the algorithm. $\boldsymbol{\beta}_j$ in (4.3.22) can be calculated using previous values of $\boldsymbol{\beta}_j$'s. Thus, (4.3.22) can be written as

$$\boldsymbol{\beta}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{\left(p_{ij}^{(r-1)}\right)^2 \mathbf{K}^{-1} \mathbf{K}_i}{\sqrt{\mathbf{K}_{ii} - 2\left(\boldsymbol{\beta}_j^{(r-1)}\right)^T \mathbf{K}_i + \left(\boldsymbol{\beta}_j^{(r-1)}\right)^T \mathbf{K} \boldsymbol{\beta}_j^{(r-1)}}}}{\sum_{i=1}^{N} \frac{\left(p_{ij}^{(r-1)}\right)^2}{\sqrt{\mathbf{K}_{ii} - 2\left(\boldsymbol{\beta}_j^{(r-1)}\right)^T \mathbf{K}_i + \left(\boldsymbol{\beta}_j^{(r-1)}\right)^T \mathbf{K} \boldsymbol{\beta}_j^{(r-1)}}}}, \tag{4.3.23}$$

where r is the current iteration.

Using the updated $\boldsymbol{\beta}_j$'s and substituting them in (4.3.15), p_{ij}'s can be updated as

$$p_{ij}^{(r)} = \frac{\frac{1}{\sqrt{\mathbf{K}_{ii} - 2\left(\boldsymbol{\beta}_j^{(r)}\right)^T \mathbf{K}_i + \left(\boldsymbol{\beta}_j^{(r)}\right)^T \mathbf{K} \boldsymbol{\beta}_j^{(r)}}}}{\sum_{t=1}^{T} \frac{1}{\sqrt{\mathbf{K}_{ii} - 2\left(\boldsymbol{\beta}_t^{(r)}\right)^T \mathbf{K}_i + \left(\boldsymbol{\beta}_t^{(r)}\right)^T \mathbf{K} \boldsymbol{\beta}_t^{(r)}}}}. \tag{4.3.24}$$

The pseudocode of **KPD Algorithm** is given in Algorithm 1 below.

Algorithm 1: KPD Algorithm

Input : data set **X**, coefficient matrix **B**, number of clusters T, kernel function k, stopping criterion ϵ

Output: probability matrix **P**

1 *Initialize* **B** to a random coefficient matrix and *Calculate* kernel matrix **K**

2 *Calculate* $p_{ij}^{(0)}$'s in **P** matrix as in (4.3.24).

3 *Set* $r = 0$.

4 **while** $\sum_{j=1}^{T} |\boldsymbol{\beta_j}^{(r)} - \boldsymbol{\beta_j}^{(r-1)}| > \epsilon$ **do**

5 $\quad r = r + 1$

6 $\quad$ *Update* $\boldsymbol{\beta_j}^{(r)}$ as in (4.3.23)

7 $\quad$ *Update* $p_{ij}^{(r)}$ as in (4.3.24)

8 **end**

4.3.1.2 Cluster Centers in Input Space

Although **KPD Algorithm** defines the centers in the *kernel space*, the centers in (4.3.5) cannot be calculated explicitly due to the ϕ function so the centers are updated implicitly through the iterations. The centers are required only when the distance between the data points and the centers are calculated. We embedded the center equation in the distance function, and using kernel trick we obtained the corresponding distances. Then the algorithm progresses without calculating the centers, only with updating the probabilities p_{ij}'s and center coefficient vectors $\boldsymbol{\beta}_j$'s (see steps (6) and (7) in Algorithm 1). In fact, the center updates are made in the *kernel space* implicitly because $\boldsymbol{\beta}_j$'s are the coefficients of the kernel centers.

In the case that the cluster centers in the *input space* needs to be known, it leads to the *preimage* problem. The inverse image of kernel center $\mathbf{c}_j$ in the *input space* is called as *preimage*, and is denoted by $\mathbf{c}_j^I$. To get a center in the *input space*, one should find the inverse mapping of it, which is $\phi^{-1}(\mathbf{c}_j)$. If ϕ is invertible, then $\phi^{-1}(\mathbf{c}_j) = \mathbf{c}_j^I$.

Note that the mapping function is nonlinear. Some kernel functions such as poly-

nomial kernels with odd exponent term or sigmoid kernel are invertible. However, for many kernel functions, inverse may not exist since some points on the span of $\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), ..., \phi(\mathbf{x}_N)$ may not correspond to the image of a single point in the *input space*. Figure 4.2 provides an example. When the data points in $\mathbf{X}$ are mapped into the *kernel space*, their image becomes dark gray-colored region, shown by $\phi(\mathbf{X})$. However, the point $\mathbf{c}_j$, which can be written in terms of mapped data points, is not an image of any point in $\mathbf{X}$. Therefore, it does not have an exact *preimage* since we cannot find a point $\mathbf{c}_j^I$ where $\phi(\mathbf{c}_j^I) = \mathbf{c}_j$.

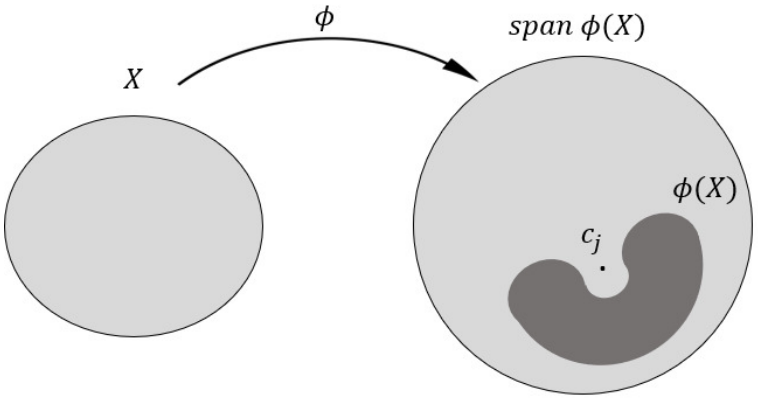


Figure 4.2: Exact Preimage Problem

For instance, Gaussian kernel function is not invertible so no exact *preimage* can be found. In that case, the approximate *preimage* of a cluster center can be obtained.

Let $\tilde{\mathbf{c}}_j \in \mathbf{X}$, and $\boldsymbol{\psi} = \alpha\, \phi(\tilde{\mathbf{c}}_j)$ tries to approximate $\mathbf{c}_j$ so $\tilde{\mathbf{c}}_j$ is called the approximate *preimage*. [35] suggests to minimize the distance between $\mathbf{c}_j$ and the orthogonal projection of it onto span $\phi(\tilde{\mathbf{c}}_j)$, $\alpha\, \phi(\tilde{\mathbf{c}}_j)$, instead of minimizing the squared distance between $\mathbf{c}_j$ and $\boldsymbol{\psi}$, where

$$\|\mathbf{c}_j - \boldsymbol{\psi}\| = \sum_{i=1}^{N}\sum_{l=1}^{N} \beta_{ij}\, \beta_{lj}\, k(\mathbf{x}_i, \mathbf{x}_l) - 2\sum_{i=1}^{N} \beta_{ij}\, \alpha\, k(\mathbf{x}_i, \tilde{\mathbf{c}}_j) + \alpha^2\, k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j)\,. \quad (4.3.25)$$

Figure 4.3 describes the orthogonal projection.

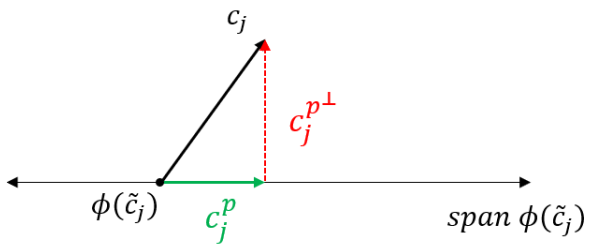


Figure 4.3: Orthogonal Projection of $\mathbf{c}_j$ onto span $\phi(\tilde{\mathbf{c}}_j)$

The orthogonal projection of $\mathbf{c}_j$, shown by $\mathbf{c}_j^p$, is $\dfrac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} \phi(\tilde{\mathbf{c}}_j)$. Then the objective is

$$\min \qquad \left\| \mathbf{c}_j - \frac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} \phi(\tilde{\mathbf{c}}_j) \right\|^2 . \tag{4.3.26}$$

This objective minimizes $\|\mathbf{c}_j^{p^\perp}\|^2$, where $\mathbf{c}_j^{p^\perp} = \mathbf{c}_j - \mathbf{c}_j^p$. From Pythagorean theorem, we know that $\|\mathbf{c}_j^{p^\perp}\|^2 = \|\mathbf{c}_j\|^2 - \|\mathbf{c}_j^p\|^2$. Since $\|\mathbf{c}_j^p\|^2 = \dfrac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle^2}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle}$, (4.3.26) becomes

$$\min \qquad \|\mathbf{c}_j\|^2 - \frac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle^2}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} . \tag{4.3.27}$$

Note that minimizing (4.3.27) is also equal to

$$\max \qquad \frac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle^2}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} . \tag{4.3.28}$$

If we want to minimize (4.3.25), then the optimal α value is obtained by taking the derivative with respect to α and making it equal to 0, which gives

$$2\,\alpha k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j) - 2 \sum_{i=1}^{N} \beta_{ij}\, k(\tilde{\mathbf{c}}_j, \mathbf{x}_i) = 0\,.$$

Then

$$\alpha = \frac{\sum_{i=1}^{N} \beta_{ij}\, k(\tilde{\mathbf{c}}_j, \mathbf{x}_i)}{k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j)} = \frac{\sum_{i=1}^{N} \beta_{ij}\, \langle \phi(\tilde{\mathbf{c}}_j), \phi(\mathbf{x}_i) \rangle}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} = \frac{\langle \phi(\tilde{\mathbf{c}}_j), \mathbf{c}_j \rangle}{\langle \phi(\tilde{\mathbf{c}}_j), \phi(\tilde{\mathbf{c}}_j) \rangle} . \tag{4.3.29}$$

If α is chosen as in (4.3.29), then minimizing (4.3.25) is equal to maximizing (4.3.28). We prefer the former objective since it is easier to work with. If we want to minimize

(4.3.25) for each cluster, we can obtain an objective that is the sum of the squared distances between $\tilde{\mathbf{c}}_j$ and $\mathbf{c}_j$, which is

$$\min_{\tilde{\mathbf{c}}_j} \qquad V = \sum_{j=1}^{T} \|\phi(\tilde{\mathbf{c}}_j) - \mathbf{c}_j\|^2 \tag{4.3.30}$$

since minimizing each cluster center is independent of each other.

The function V in (4.3.30) can be written as

$$\sum_{j=1}^{T} \|\phi(\tilde{\mathbf{c}}_j) - \mathbf{c}_j\| = \sum_{j=1}^{T} \sqrt{k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j) - 2\,\phi(\tilde{\mathbf{c}}_j)\,\mathbf{c}_j + \mathbf{c}_j^T \mathbf{c}_j)}\,. \tag{4.3.31}$$

Then $\mathbf{c}_j$'s can be expressed as in (4.3.5), so we substitute them in (4.3.31). Thus, the multiplication of two ϕ functions will be obtained, and they can be written as a function of K using the kernel trick. That is, (4.3.31) becomes

$$\sum_{j=1}^{T} \left(k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j) - 2 \frac{\sum_{i=1}^{N} \frac{p_{ij}^2\, k(\mathbf{x}_i, \tilde{\mathbf{c}}_j)}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} + \sum_{l=1}^{N} \sum_{k=1}^{N} \frac{\frac{p_{lj}^2\, p_{kj}^2\, k(\mathbf{x}_l, \mathbf{x}_k)}{\|\phi(\mathbf{x}_l) - \mathbf{c}_j\| \|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}}{\left(\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} \right) \left(\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} \right)} \right)^{1/2}. \tag{4.3.32}$$

To find the optimum $\tilde{\mathbf{c}}_j$, one can take the derivative of (4.3.31) with respect to $\tilde{c}_j$ and make it equal to 0. However, to take the derivative, prior information about the kernel function must be known. Below, Gaussian and polynomial kernel functions are considered to calculate the centers.

- **Gaussian Kernel**

When the kernel function is chosen as Gaussian kernel, (4.3.32) becomes

$$\sum_{j=1}^{T} \sqrt{k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j) - 2\frac{\sum_{i=1}^{N} \frac{p_{ij}^2\, e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j\|^2/\sigma^2}}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} + \sum_{l=1}^{N}\sum_{k=1}^{N} \frac{\frac{p_{lj}^2\, p_{kj}^2\, e^{-\|\mathbf{x}_l - \mathbf{x}_k\|^2/\sigma^2}}{\|\phi(\mathbf{x}_l) - \mathbf{c}_j\|\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}}{\left(\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}\right)^2}} \tag{4.3.33}$$

Since in Gaussian kernel the distance between the object and itself is equal to 1, $k(\tilde{\mathbf{c}}_j, \tilde{\mathbf{c}}_j) = 1$ in Equation (4.3.33). Then taking the derivative of (4.3.33) with respect to $\tilde{\mathbf{c}}_j$ and equalizing it to 0 gives

$$\begin{aligned}\frac{\partial V}{\partial \tilde{\mathbf{c}}_j} &= \frac{1}{2}\, \frac{-2}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} \sum_{i=1}^{N} \left(\frac{p_{ij}^2\, e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j\|^2/\sigma^2}}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} \frac{2\,(\mathbf{x}_i - \tilde{\mathbf{c}}_j)}{\sigma^2} \right) \frac{1}{\|\phi(\tilde{\mathbf{c}}_j) - \mathbf{c}_j\|} \\ &= \mathbf{0}\,.\end{aligned}$$

Then

$$\tilde{\mathbf{c}}_j = \frac{\sum_{i=1}^{N} \frac{p_{ij}^2\, \mathbf{x}_i}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j\|^2/\sigma^2}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j\|^2/\sigma^2}} \tag{4.3.34}$$

for Gaussian kernel.

- **Polynomial Kernel**

If the Polynomial kernel is considered, now (4.3.32) becomes

$$\sum_{j=1}^{T} \left((\tilde{\mathbf{c}}_j^T\, \tilde{\mathbf{c}}_j + a)^b - 2\, \frac{\sum_{i=1}^{N} \frac{p_{ij}^2\, (\tilde{\mathbf{c}}_j^T\, \mathbf{x}_i + a)^b}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} + \sum_{l=1}^{N}\sum_{k=1}^{N} \frac{\frac{p_{lj}^2\, p_{kj}^2\, (\mathbf{x}_l^T\, \mathbf{x}_k + a)^b}{\|\phi(\mathbf{x}_l) - \mathbf{c}_j\|\|\phi(\mathbf{x}_k) - \mathbf{c}_j\|}}{\left(\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}\right)\left(\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}\right)} \right)^{1/2}, \tag{4.3.35}$$

where a and b are the predefined parameters of the Polynomial kernel.

The derivative of (4.3.35) with respect to $\tilde{\mathbf{c}}_j$ is

$$\frac{\partial V}{\partial \tilde{\mathbf{c}}_j} = \frac{1}{2}\left(2\,b\,\tilde{\mathbf{c}}_j(\tilde{\mathbf{c}}_j^T\,\tilde{\mathbf{c}}_j + a)^{b-1} - 2\frac{\sum_{i=1}^{N} \frac{p_{ij}^2\, b\,(\tilde{\mathbf{c}}_j^T\,\mathbf{x}_i + a)^{b-1}\,\mathbf{x}_i}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} \right) \frac{1}{\|\phi(\tilde{\mathbf{c}}_j) - \mathbf{c}_j\|}$$

$$= \mathbf{0}\,.$$

Then

$$\tilde{\mathbf{c}}_j = \frac{\sum_{i=1}^{N} \left(\frac{p_{ij}^2\,(\tilde{\mathbf{c}}_j^T\,\mathbf{x}_i + a)^{b-1}}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|} \right) \mathbf{x}_i}{(\tilde{\mathbf{c}}_j^T\,\tilde{\mathbf{c}}_j + a)^{b-1} \sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \mathbf{c}_j\|}} \tag{4.3.36}$$

for Polynomial kernel.

4.3.1.3 Center Update

$\tilde{\mathbf{c}}_j$ values can be calculated using (4.3.34) and (4.3.36) but these equations contain $\mathbf{c}_j$'s in the denominator terms. Therefore, $\mathbf{c}_j$'s can be approximated with the image of $\tilde{\mathbf{c}}_j$'s from the previous iteration, so using kernel trick we obtain

$$\tilde{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2\,\mathbf{x}_i}{\sqrt{2 - 2\,k\left(\mathbf{x}_i, \tilde{\mathbf{c}}_j^{(r-1)}\right)}} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2}{\sqrt{2 - 2\,k\left(\mathbf{x}_i, \tilde{\mathbf{c}}_j^{(r-1)}\right)}} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}} \tag{4.3.37}$$

for Gaussian kernel. When function k is replaced with Gaussian kernel, (4.3.37) becomes

$$\tilde{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2\,\mathbf{x}_i}{\sqrt{2 - 2\,e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2}{\sqrt{2 - 2\,e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}} e^{-\|\mathbf{x}_i - \tilde{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}\,. \tag{4.3.38}$$

For polynomial kernel, $\mathbf{c}_j$'s are approximated and (4.3.36) is obtained as

$$\tilde{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \left((\tilde{\mathbf{c}}_j^{(r-1)})^T \mathbf{x}_i + a\right)^{b-1}}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2k(\mathbf{x}_i, \tilde{\mathbf{c}}_j^{(r-1)}) + k(\tilde{\mathbf{c}}_j^{(r-1)}, \tilde{\mathbf{c}}_j^{(r-1)})}} \mathbf{x}_i}{(\tilde{\mathbf{c}}_j^{(r-1)})^T \tilde{\mathbf{c}}_j^{(r-1)} + a)^{b-1} \sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2k(\mathbf{x}_i, \tilde{\mathbf{c}}_j^{(r-1)}) + k(\tilde{\mathbf{c}}_j^{(r-1)}, \tilde{\mathbf{c}}_j^{(r-1)})}}} \tag{4.3.39}$$

using kernel trick. Writing polynomial kernel function explicitly in (4.3.39) would give

$$\tilde{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \left((\tilde{\mathbf{c}}_j^{(r-1)})^T \mathbf{x}_i + a\right)^{b-1}}{\sqrt{\left(\mathbf{x}_i^T \mathbf{x}_i + a\right)^b - 2\left(\mathbf{x}_i^T \tilde{\mathbf{c}}_j^{(r-1)} + a\right)^b + \left(\tilde{\mathbf{c}}_j^{(r-1)T} \tilde{\mathbf{c}}_j^{(r-1)} + a\right)^b}} \mathbf{x}_i}{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \left(\tilde{\mathbf{c}}_j^{(r-1)}\right)^T \tilde{\mathbf{c}}_j^{(r-1)} + a)^{b-1}}{\sqrt{\left(\mathbf{x}_i^T \mathbf{x}_i + a\right)^b - 2\left(\mathbf{x}_i^T \tilde{\mathbf{c}}_j^{(r-1)} + a\right)^b + \left(\tilde{\mathbf{c}}_j^{(r-1)T} \tilde{\mathbf{c}}_j^{(r-1)} + a\right)^b}}} \cdots \tag{4.3.40}$$

The centers in any iteration will be updated as (4.3.38) for Gaussian kernel and (4.3.40) for polynomial kernel.

A new version of **KPD Algorithm** where the *input space* centers can be derived is called **KPD-C Algorithm**, and Algorithm 2 provides the pseudocode of it.

4.3.2 Kernel Pd-clustering in Feature Space

KPD Algorithm defines the cluster centers in the *kernel space*. Since the centers in the *kernel space* cannot be calculated, they are updated implicitly. In addition, the algorithm proposes a method to find the approximate centers in the *input space*. On the other hand, Kernel Pd-clustering in Feature Space Algorithm (**KPD-F Algorithm**) defines and updates the centers in the *input space*.

Let $\hat{\mathbf{c}}_j$ be the cluster centers in the *input space*. Then $\phi(\hat{\mathbf{c}}_j)$ gives the image of them in the *kernel space*. To calculate the distance between each data point and their corresponding clusters in *kernel space*, both data points and the cluster centers are required to be mapped into *kernel space* beforehand as shown in Figure 4.4.

Algorithm 2: KPD-C Algorithm

Input : data set **X**, coefficient matrix **B**, number of clusters T, kernel function k, stopping criterion ϵ, stopping criterion for centers ϵ_c

Output: probability matrix **P**

Initialize **B** to a random coefficient matrix and *Calculate* kernel matrix **K**

Calculate $p_{ij}^{(0)}$'s in **P** matrix as in (4.3.24).

Set $r = 0$.

while $\sum_{j=1}^{T} |\boldsymbol{\beta_j}^{(r)} - \boldsymbol{\beta_j}^{(r-1)}| > \epsilon$ **do**

  $r = r + 1$

  Update $\boldsymbol{\beta_j}^{(r)}$ as in (4.3.23)

  Update $p_{ij}^{(r)}$ as in (4.3.24)

end

Set $r = 0$. Use optimal probability matrix **P***.

Initialize $\tilde{\mathbf{c}}_j^{(0)}$ as a random center of cluster j for $j = 1, ...T$.

while $\sum_{j=1}^{T} |\tilde{\boldsymbol{c}}_j^{(r)} - \tilde{\boldsymbol{c}}_j^{(r-1)}| > \epsilon_c$ **do**

  $r = r + 1$

  Update $\tilde{\mathbf{c}}_j^{(r)}$ as in (4.3.38) if kernel function is Gaussian, and update as in (4.3.40) if kernel is chosen as polynomial

end

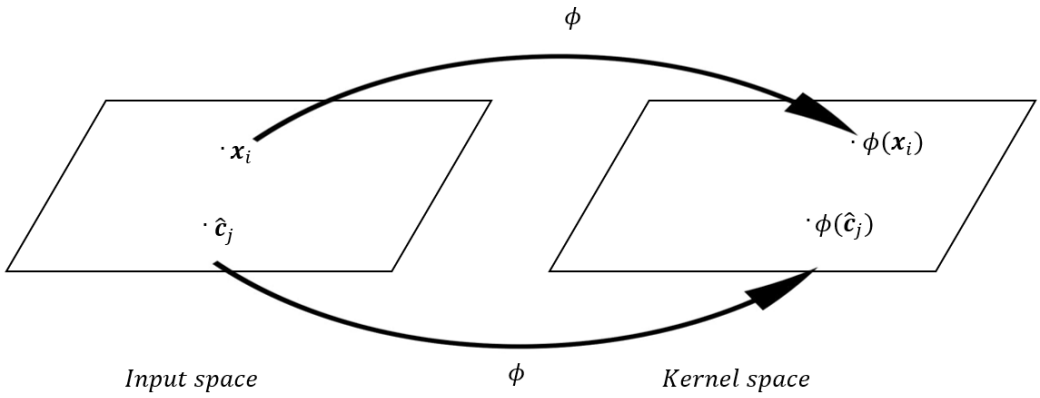


Figure 4.4: Kernel Pd-clustering in Feature Space in Input and Kernel Space

Then the optimization problem becomes

$$\begin{aligned} \min \quad & \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \left\|\phi\left(\mathbf{x}_i\right) - \phi\left(\hat{\mathbf{c}}_j\right)\right\| & \\ \text{s.t.} \quad & \sum_{j=1}^{T} p_{ij} = 1 & \forall i \\ & p_{ij} \geq 0 & \forall i, j \end{aligned}$$

where $\hat{\mathbf{c}}_j$ represents cluster centers in input space. Let $\mathbf{\Phi_c}$ be a $T \times 1$ vector containing the images of *input space* centers. That is, $\mathbf{\Phi_c} = \left[\phi(\hat{\mathbf{c}}_1) \quad \phi(\hat{\mathbf{c}}_2) \quad \dots \quad \phi(\hat{\mathbf{c}}_T)\right]^T$. Then the Lagrangian becomes

$$L(\mathbf{P}, \mathbf{\Phi_c}, \mathbf{\Lambda}) = \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \left\|\phi\left(\mathbf{x}_i\right) - \phi\left(\hat{\mathbf{c}}_j\right)\right\| - \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{T} p_{ij} - 1\right). \quad (4.3.41)$$

Taking the derivative with respect to p_{ij} we obtain

$$p_{ij} = \frac{\dfrac{1}{\left\|\phi\left(\mathbf{x}_i\right) - \phi\left(\hat{\mathbf{c}}_t\right)\right\|}}{\sum\limits_{t=1}^{T} \dfrac{1}{\left\|\phi(\mathbf{x}_i) - \phi\left(\hat{\mathbf{c}}_t\right)\right\|}} \cdot \quad (4.3.42)$$

Given p_{ij}'s, if we take the derivative of (4.3.41) with respect to $\phi(\hat{\mathbf{c}}_j)$, we obtain

$$\phi(\hat{\mathbf{c}}_j) = \frac{\sum_{i=1}^{N} \frac{p_{ij}^2 \, \phi(\mathbf{x}_i)}{\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{c}}_j)\|}}{\sum_{i=1}^{N} \frac{p_{ij}^2}{\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{c}}_j)\|}}. \tag{4.3.43}$$

In (4.3.43), $\phi(\hat{\mathbf{c}}_j)$ depends on $\phi(\mathbf{x}_i)$'s, which cannot be derived. Therefore, the derivative of (4.3.41) with respect to $\hat{\mathbf{c}}_j$ can be calculated only when the kernel function is known since the distance term can be rewritten using kernel trick, and $\hat{\mathbf{c}}_j$ becomes the input of kernel function. We consider two kernel functions which are Gaussian and polynomial kernels.

- **Gaussian Kernel**

  We observe $\|\phi(\mathbf{x}_i) - \phi(\hat{\mathbf{c}}_j)\|$ can be written as

  $$\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j) + k(\hat{\mathbf{c}}_j, \hat{\mathbf{c}}_j)} = \sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)}. \tag{4.3.44}$$

  for Gaussian kernel. If we substitute the distance term in (4.3.41) with (4.3.44), then L becomes the function of p_{ij}'s and $\hat{\mathbf{c}}_j$'s. To take the gradient of L, we need to know kernel function. In the case of Gaussian kernel, Lagrangian becomes

  $$L(\mathbf{P}, \mathbf{C}, \mathbf{\Lambda}) = \sum_{i=1}^{N} \sum_{j=1}^{T} p_{ij}^2 \sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)} - \sum_{i=1}^{N} \lambda_i \left(\sum_{j=1}^{T} p_{ij} - 1 \right). \tag{4.3.45}$$

  Then for the given p_{ij}'s the derivative of (4.3.45) with respect to $\hat{\mathbf{c}}_j$ is

  $$\frac{\partial L}{\partial \hat{\mathbf{c}}_j} = \frac{1}{2} \sum_{i=1}^{N} p_{ij}^2 \frac{1}{\sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)}} (-2) \, e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j\|^2/\sigma^2} \, \frac{2(\mathbf{x}_i - \hat{\mathbf{c}}_j)}{\sigma^2} = 0$$

  $$\implies \sum_{i=1}^{N} p_{ij}^2 \frac{e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j\|^2/\sigma^2}}{\sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)}} (\mathbf{x}_i - \hat{\mathbf{c}}_j) = 0.$$

  Therefore, we obtain $\hat{\mathbf{c}}_j$ as

  $$\hat{\mathbf{c}}_j = \frac{\sum_{i=1}^{N} \left(p_{ij}^2 \frac{e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j\|^2/\sigma^2}}{\sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)}} \right) \mathbf{x}_i}{\sum_{i=1}^{N} p_{ij}^2 \frac{e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j\|^2/\sigma^2}}{\sqrt{2 - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j)}}}. \tag{4.3.46}$$

- **Polynomial Kernel**

  When the kernel function is chosen as polynomial, $\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\|$ is calculated as

$$\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\| = \sqrt{(\mathbf{x}_i^T\mathbf{x}_i+a)^b - 2(\hat{\mathbf{c}}_j^T\mathbf{x}_i+a)^b + (\hat{\mathbf{c}}_j^T\hat{\mathbf{c}}_j+a)^b}.$$

  Lagrangian becomes

$$L(\mathbf{P},\mathbf{C},\mathbf{\Lambda}) = \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2 \sqrt{(\mathbf{x}_i^T\mathbf{x}_i+a)^b - 2(\hat{\mathbf{c}}_j^T\mathbf{x}_i+a)^b + (\hat{\mathbf{c}}_j^T\hat{\mathbf{c}}_j+a)^b} - \sum_{i=1}^{N}\lambda_i\left(\sum_{j=1}^{T} p_{ij} - 1\right). \tag{4.3.47}$$

  For given p_{ij}'s, the derivative of (4.3.47) with respect to $\hat{\mathbf{c}}_j$ becomes

$$\frac{\partial L}{\partial \hat{\mathbf{c}}_j} = \frac{1}{2}\sum_{i=1}^{N} p_{ij}^2 \frac{\left(-2\,b\,\mathbf{x}_i\,(\hat{\mathbf{c}}_j^T\,\mathbf{x}_i+a)^{b-1} + 2\,b\,\hat{\mathbf{c}}_j\,(\hat{\mathbf{c}}_j^T\,\hat{\mathbf{c}}_j+a)^{b-1}\right)}{\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\|} = 0\,.$$

  Then

$$\sum_{i=1}^{N}\frac{p_{ij}^2}{\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\|}\left(\hat{\mathbf{c}}_j\,(\hat{\mathbf{c}}_j^T\,\hat{\mathbf{c}}_j+a)^{b-1} - \mathbf{x}_i\,(\hat{\mathbf{c}}_j^T\,\mathbf{x}_i+a)^{b-1}\right) = 0\,,$$

  which gives

$$\hat{\mathbf{c}}_j = \frac{\sum_{i=1}^{N}\frac{p_{ij}^2}{\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\|}(\hat{\mathbf{c}}_j^T\,\mathbf{x}_i+a)^{b-1}\,\mathbf{x}_i}{(\hat{\mathbf{c}}_j^T\,\hat{\mathbf{c}}_j+a)^{b-1}\sum_{i=1}^{N}\frac{p_{ij}^2}{\|\phi(\mathbf{x}_i)-\phi(\hat{\mathbf{c}}_j)\|}}. \tag{4.3.48}$$

4.3.2.1 Center Update

Since the denominator terms in center equations for both Gaussian and polynomial kernels have $\hat{\mathbf{c}}_j$'s, previous center values should be used when they are updated. Thus, (4.3.46) is rearranged as

$$\hat{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N}\left((p_{ij}^{(r-1)})^2\frac{e^{-\|\mathbf{x}_i-\hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sqrt{2-2k(\mathbf{x}_i,\hat{\mathbf{c}}_j^{(r-1)})}}\right)\mathbf{x}_i}{\sum_{i=1}^{N}(p_{ij}^{(r-1)})^2\frac{e^{-\|\mathbf{x}_i-\hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sqrt{2-2k(\mathbf{x}_i,\hat{\mathbf{c}}_j^{(r-1)})}}} \tag{4.3.49}$$

for Gaussian kernel. If function k is replaced with Gaussian kernel, then (4.3.49) becomes

$$\hat{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \left((p_{ij}^{(r-1)})^2 \frac{e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sqrt{2 - 2\, e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}} \right) \mathbf{x}_i}{\sum_{i=1}^{N} (p_{ij}^{(r-1)})^2 \frac{e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}{\sqrt{2 - 2\, e^{-\|\mathbf{x}_i - \hat{\mathbf{c}}_j^{(r-1)}\|^2/\sigma^2}}}}. \tag{4.3.50}$$

In the case of polynomial kernel, (4.3.48) can be rewritten as

$$\hat{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \, ((\hat{\mathbf{c}}_j^{(r-1)})^T \mathbf{x}_i + a)^{b-1}}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j^{(r-1)}) + k(\hat{\mathbf{c}}_j^{(r-1)}, \hat{\mathbf{c}}_j^{(r-1)})}} \mathbf{x}_i}{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \, (\hat{\mathbf{c}}_j^{(r-1)})^T \hat{\mathbf{c}}_j^{(r-1)} + a)^{b-1}}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) - 2k(\mathbf{x}_i, \hat{\mathbf{c}}_j^{(r-1)}) + k(\hat{\mathbf{c}}_j^{(r-1)}, \hat{\mathbf{c}}_j^{(r-1)})}}},$$

which gives

$$\hat{\mathbf{c}}_j^{(r)} = \frac{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \, ((\hat{\mathbf{c}}_j^{(r-1)})^T \mathbf{x}_i + a)^{b-1}}{\sqrt{(\mathbf{x}_i^T \mathbf{x}_i + a)^b - 2\left(\mathbf{x}_i^T \phi(\hat{\mathbf{c}}_j^{(r-1)}) + a\right)^b + \left(\phi(\hat{\mathbf{c}}_j^{(r-1)})^T \phi(\hat{\mathbf{c}}_j^{(r-1)}) + a\right)^b}} \mathbf{x}_i}{\sum_{i=1}^{N} \frac{(p_{ij}^{(r-1)})^2 \, (\hat{\mathbf{c}}_j^{(r-1)})^T \hat{\mathbf{c}}_j^{(r-1)} + a)^{b-1}}{\sqrt{(\mathbf{x}_i^T \mathbf{x}_i + a)^b - 2\left(\mathbf{x}_i^T \phi(\hat{\mathbf{c}}_j^{(r-1)}) + a\right)^b + \left(\phi(\hat{\mathbf{c}}_j^{(r-1)})^T \phi(\hat{\mathbf{c}}_j^{(r-1)}) + a\right)^b}}}. \tag{4.3.51}$$

4.3.2.2 Probability Update

Upon completing the center updates in the *input space*, p_{ij}'s are updated using $\hat{\mathbf{c}}_j^{(r)}$'s. Then (4.3.42) becomes

$$p_{ij}^{(r)} = \frac{\frac{1}{\left\| \phi(\mathbf{x}_i) - \phi\left(\hat{\mathbf{c}}_t^{(r)}\right) \right\|}}{\sum_{t=1}^{T} \frac{1}{\left\| \phi(\mathbf{x}_i) - \phi\left(\hat{\mathbf{c}}_t^{(r)}\right) \right\|}}.$$

Distance terms in the denominator terms of the probability update is a function of k, and it refers to

$$p_{ij}^{(r)} = \frac{\dfrac{1}{\sqrt{k(\mathbf{x}_i,\mathbf{x}_i) - 2k(\mathbf{x}_i,\hat{\mathbf{c}}_j^{(r)}) + k(\hat{\mathbf{c}}_j^{(r)},\hat{\mathbf{c}}_j^{(r)})}}}{\sum_{t=1}^{T} \dfrac{1}{\sqrt{k(\mathbf{x}_i,\mathbf{x}_i) - 2k(\mathbf{x}_i,\hat{\mathbf{c}}_j^{(r)}) + k(\hat{\mathbf{c}}_j^{(r)},\hat{\mathbf{c}}_j^{(r)})}}}. \tag{4.3.52}$$

4.3.2.3 KPD-F Algorithm

This algorithm defines and updates the centers in the *input space*. Since they are introduced in the *input space*, they are known explicitly. After the center update, probabilities are updated accordingly. Pseudocode of **KPD-F Algorithm** is given in Algorithm 3.

Algorithm 3: KPD-F Algorithm

Input : data set $\mathbf{X}$, number of clusters T, kernel function k, stopping criterion ϵ

Output: probability matrix $\mathbf{P}$

Initialize $\hat{\mathbf{c}}_j^{(0)}$ as a random center of cluster j for $j = 1, ..., T$ and *Calculate* $k(\mathbf{x}_i, \mathbf{x}_l)$ for $i, l \in 1, ..., N$.

Calculate $p_{ij}^{(0)}$'s in $\mathbf{P}$ matrix as in (4.3.52).

Set $r = 0$.

while $\sum_{j=1}^{T} |\hat{\boldsymbol{c}}_j^{(r)} - \hat{\boldsymbol{c}}_j^{(r-1)}| > \epsilon$ **do**

  $r = r + 1$

  Update $\hat{\mathbf{c}}_j^{(r)}$ as in (4.3.50) if kernel function is Gaussian, and update as in (4.3.51) if kernel is chosen as polynomial

  Calculate $k(\mathbf{x}_i, \hat{\mathbf{c}}_j^{(r)})$ for $i \in 1, ..., N, j \in 1, ..., T$

  Update $p_{ij}^{(r)}$ as in (4.3.52)

end

CHAPTER 5

KERNEL PROBABILISTIC DISTANCE CLUSTERING WITH MAHALANOBIS DISTANCE

We consider Euclidean norm in the algorithms explained in Section 4.3. In this chapter, following the ideas in [36], we study the statistical distance (i.e., Mahalanobis distance) in the kernel algorithms, where the correlation between the data features is considered. First, we introduce two Kernel Mahalanobis distance functions. Later, Kernel probabilistic distance clustering with Mahalanobis distance will be introduced.

5.1 Kernel Mahalanobis Distance

Consider we have an $m \times n$ data set $\mathbf{X}$, where n shows the number of data points and m is that of features in the *input space*. Therefore, each data point $\mathbf{x}_i$ is an m-dimensional column vector, $i = 1, ..., n$. Function ϕ represents the mapping from original space to Hilbert Space $\mathscr{H}$ (or *kernel space*), i.e. $\phi : X \rightarrow \mathscr{H}$. In this mapping, the dimension of the vectors is changed to s. Therefore, $\phi(\mathbf{x}_i)$ becomes $s \times 1$ vector.

Let $\mathbf{\Phi}$ contains the mapping of each data point $\mathbf{x}_i$. Then $\mathbf{\Phi}$ is an $s \times n$ matrix shown as $\mathbf{\Phi} = [\phi(\mathbf{x}_1)...\phi(\mathbf{x}_n)]_{s \times n}$. The mean of $\phi(\mathbf{x}_i)$'s are calculated as

$$\boldsymbol{\phi}_\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(\mathbf{x}_i) = \frac{1}{n}\mathbf{\Phi}\mathbf{1}_n\,, \tag{5.1.1}$$

where $\mathbf{1}_n$ is a column vector of 1's. If data point $\phi(\mathbf{x}_i)$ is centered with $\boldsymbol{\phi}_\mu$, it is denoted by

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \boldsymbol{\phi}_\mu\,. \tag{5.1.2}$$

Using (5.1.1), we can write (5.1.2) as

$$\tilde{\phi}(\mathbf{x}_i) = \phi(\mathbf{x}_i) - \frac{1}{n}\mathbf{\Phi 1}_n \,. \tag{5.1.3}$$

Let $\tilde{\mathbf{\Phi}}$ represent the matrix of centered data points $\mathbf{x}_i$, $i = 1, ..., n$. Then

$$\tilde{\mathbf{\Phi}} = \left[\tilde{\phi}(\mathbf{x}_1)...\tilde{\phi}(\mathbf{x}_n)\right] = \mathbf{\Phi} - \boldsymbol{\phi}_\mu \mathbf{1}_n^T \,. \tag{5.1.4}$$

We can substitute (5.1.1) in (5.1.4) for $\boldsymbol{\phi}_\mu$and obtain

$$\tilde{\mathbf{\Phi}} = \mathbf{\Phi} - \frac{1}{n}\mathbf{\Phi 1}_n \mathbf{1}_n^T = \mathbf{\Phi}\left[\mathbf{I}_{n\times n} - \frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T\right] \,. \tag{5.1.5}$$

Then (5.1.5) can be written as

$$\tilde{\mathbf{\Phi}} = \mathbf{\Phi H} \,, \tag{5.1.6}$$

where $\mathbf{H}$ is an $n \times n$ centering matrix shown as

$$\mathbf{H} = \mathbf{I}_{n\times n} - \frac{1}{n}\mathbf{1}_n \mathbf{1}_n^T \,.$$

$\mathbf{H}$ matrix has some properties. The transpose and square of $\mathbf{H}$ is equal to itself. That is, $\mathbf{H} = \mathbf{H}^T = \mathbf{H}^2$.

The covariance operator in the Hilbert Space, shown as $\mathbf{C} : \mathscr{H} \to \mathscr{H}$, operates on $\phi(\mathbf{x}) \in \mathscr{H}$ as

$$\mathbf{C}\phi(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\left(\phi(\mathbf{x}_i) - \boldsymbol{\phi}_\mu\right)\langle\phi(\mathbf{x}_i) - \boldsymbol{\phi}_\mu, \phi(\mathbf{x}_i)\rangle. \tag{5.1.7}$$

We know that $\left(\phi(\mathbf{x}_i) - \boldsymbol{\phi}_\mu\right)$ is equal to $\tilde{\phi}(\mathbf{x}_i)$, so (5.1.7) can be written as

$$\mathbf{C}\phi(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\tilde{\phi}(\mathbf{x}_i)\tilde{\phi}(\mathbf{x}_i)^T\phi(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T\phi(\mathbf{x}) \,.$$

Using (5.1.6), $\mathbf{C}$ becomes

$$\mathbf{C} = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T = \frac{1}{n}\mathbf{\Phi HH}^T\mathbf{\Phi}^T = \frac{1}{n}\mathbf{\Phi HH\Phi}^T = \frac{1}{n}\mathbf{\Phi H\Phi}^T \,.$$

Remember that $\tilde{\mathbf{\Phi}}$ is the matrix of centered data points in *kernel space*. Then using (5.1.6), centered kernel matrix (i.e., the matrix containing the inner products of all centered kernels of data points) is

$$\widetilde{\mathbf{K}} = \tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}} = \mathbf{H\Phi}^T\mathbf{\Phi H} = \mathbf{HKH} \,,$$

where

$$\mathbf{K} = \mathbf{\Phi}^T \mathbf{\Phi} \,. \tag{5.1.8}$$

For instance, the entry in i^{th} row and l^{th} column in $\widetilde{\mathbf{K}}$ gives the inner product of centered kernels of data points $\mathbf{x}_i$ and $\mathbf{x}_l$.

Let $\bar{\mathbf{k}}_x$ be a column vector whose i^{th} element represents the inner product of $\mathbf{x}$ and $\mathbf{x}_i$ in the *kernel space*. That is,

$$\bar{\mathbf{k}}_{\mathbf{x}} = [k(\mathbf{x}_1, \mathbf{x}), ..., k(\mathbf{x}_n, \mathbf{x})]^T = \mathbf{\Phi}^T \phi(\mathbf{x}) \,. \tag{5.1.9}$$

Then the inner product of $\tilde{\boldsymbol{\phi}}(\mathbf{x})$ with other centered kernel data points $\tilde{\boldsymbol{\phi}}(\mathbf{x}_i)$'s is

$$\tilde{\mathbf{k}}_{\mathbf{x}} = \tilde{\mathbf{\Phi}}^T \tilde{\phi}(\mathbf{x}) \,. \tag{5.1.10}$$

When (5.1.6) is substituted into (5.1.10), we obtain

$$\tilde{\mathbf{k}}_{\mathbf{x}} = (\mathbf{\Phi H})^T \tilde{\phi}(\mathbf{x}) = \mathbf{H}^T \mathbf{\Phi}^T \tilde{\phi}(\mathbf{x}) = \mathbf{H} \left(\mathbf{\Phi}^T \tilde{\phi}(\mathbf{x}) \right) \,. \tag{5.1.11}$$

Using $\boldsymbol{\phi}(\mathbf{x})$ in (5.1.2), (5.1.11) becomes

$$\tilde{\mathbf{k}}_{\mathbf{x}} = \mathbf{H} \left(\mathbf{\Phi}^T \left(\boldsymbol{\phi}(\mathbf{x}_i) - \boldsymbol{\phi}_\mu \right) \right) = \mathbf{H} \left(\mathbf{\Phi}^T \boldsymbol{\phi}(\mathbf{x}_i) - \mathbf{\Phi}^T \boldsymbol{\phi}_\mu \right) \,. \tag{5.1.12}$$

We know from (5.1.9) that $\mathbf{\Phi}^T \boldsymbol{\phi}(\mathbf{x}_i)$ in (5.1.12) is equal to $\bar{\mathbf{k}}_{\mathbf{x}}$. Moreover, by substituting (5.1.1) for $\boldsymbol{\phi}_\mu$ into (5.1.12) we get

$$\tilde{\mathbf{k}}_{\mathbf{x}} = \mathbf{H} \left(\bar{\mathbf{k}}_{\mathbf{x}} - \frac{1}{n} \mathbf{\Phi}^T \mathbf{\Phi} \mathbf{1}_n \right) \,. \tag{5.1.13}$$

Using (5.1.8) for $\mathbf{\Phi}^T \mathbf{\Phi}$, (5.1.13) becomes

$$\tilde{\mathbf{k}}_{\mathbf{x}} = \mathbf{H} \left(\bar{\mathbf{k}}_{\mathbf{x}} - \frac{1}{n} \mathbf{K} \mathbf{1}_n \right) \,. \tag{5.1.14}$$

The inner product of kernel of $\mathbf{x}$ with itself is shown as $k(\mathbf{x}, \mathbf{x})$ or $k_{\mathbf{xx}}$. That is,

$$k_{\mathbf{xx}} = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}) \,. \tag{5.1.15}$$

When $k_{\mathbf{xx}}$ is for the centered data points, we obtain

$$\tilde{k}_{\mathbf{xx}} = \tilde{\phi}(\mathbf{x})^T \tilde{\phi}(\mathbf{x}) \,. \tag{5.1.16}$$

Using (5.1.3) for $\tilde{\phi}(\mathbf{x})$ in (5.1.16) we get

$$\begin{aligned}\tilde{k}_{\mathbf{xx}} &= \left(\phi(\mathbf{x}) - \frac{1}{n}\mathbf{\Phi}\mathbf{1}_n\right)^T \left(\phi(\mathbf{x}) - \frac{1}{n}\mathbf{\Phi}\mathbf{1}_n\right) \\ &= \phi(\mathbf{x})^T\phi(\mathbf{x}) - \frac{2}{n}\mathbf{1}_n^T\mathbf{\Phi}^T\phi(\mathbf{x}) + \frac{1}{n^2}\left(\mathbf{\Phi}\mathbf{1}_n\right)^T\left(\mathbf{\Phi}\mathbf{1}_n\right) .\end{aligned}$$

Substituting (5.1.15) for $\boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x})$, (5.1.9) for $\mathbf{\Phi}^T\phi(\mathbf{x})$, and (5.1.8) for $\mathbf{\Phi}^T\mathbf{\Phi}$ gives

$$\tilde{k}_{\mathbf{xx}} = k_{\mathbf{xx}} - \frac{2}{n}\mathbf{1}_n^T\bar{\mathbf{k}}_{\mathbf{x}} + \frac{1}{n^2}\mathbf{1}_n^T\mathbf{K}\mathbf{1}_n .$$

5.2 Kernel Mahalanobis Distance for Invertible Covariance

The kernelized Mahalanobis distance is

$$d_{IC}^2(\mathbf{x}) = d_{IC}^2(\phi(\mathbf{x}); \{\boldsymbol{\phi}_\mu, \mathbf{C}\}) = (\phi(\mathbf{x}) - \boldsymbol{\phi}_\mu)^T \mathbf{C}^{-1}(\phi(\mathbf{x}) - \boldsymbol{\phi}_\mu) .$$

Therefore, the covariance matrix must be invertible to calculate the Mahalanobis distance. It restricts the dimension of $\mathscr{H}$ to a finite dimension, which is s and $s < n$. $\tilde{\mathbf{\Phi}}$ has a singular value decomposition, which is

$$\tilde{\mathbf{\Phi}} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T. \tag{5.2.1}$$

Note that $\mathbf{U} \in \mathbb{R}^{s\times s}$, $\mathbf{V} \in \mathbb{R}^{n\times n}$, and $\mathbf{\Sigma} \in \mathbb{R}^{s\times n}$, where $\mathbf{U}$ and $\mathbf{V}$ contain the eigenvectors and the eigenvalues are in the diagonals of $\mathbf{\Sigma}$ matrix. Then the covariance matrix can be written as

$$\mathbf{C} = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T . \tag{5.2.2}$$

Using (5.2.1), the covariance in (5.2.2) can be rewritten as

$$\mathbf{C} = \frac{1}{n}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\right)^T = \frac{1}{n}\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T .$$

By using the orthogonality of $\mathbf{U}$ and $\mathbf{V}$ matrices, we know that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. Then we obtain

$$\mathbf{C} = \frac{1}{n}\mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T$$

and

$$\frac{1}{n}\mathbf{C}^{-1} = \mathbf{U}\left(\mathbf{\Sigma}\mathbf{\Sigma}^T\right)^{-1}\mathbf{U}^T . \tag{5.2.3}$$

If both sides of (5.2.3) are multiplied with $\tilde{\mathbf{\Phi}}$, and (5.2.1) is substituted for $\tilde{\mathbf{\Phi}}$ on the right-hand side of the equation, we obtain

$$\frac{1}{n}\mathbf{C}^{-1}\tilde{\mathbf{\Phi}} = \mathbf{U}\left(\mathbf{\Sigma}\mathbf{\Sigma}^T\right)^{-1}\mathbf{\Sigma}\mathbf{V}^T\,. \tag{5.2.4}$$

Note that $\widetilde{\mathbf{K}} = \tilde{\mathbf{\Phi}}^T\tilde{\mathbf{\Phi}}$. Following the SVD, it is equal to

$$\widetilde{\mathbf{K}} = \mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T\,.$$

If $\tilde{\mathbf{\Phi}}$ is invertible, then $\mathbf{\Sigma}^{-1}$ exists. If $\tilde{\mathbf{\Phi}}$ is not invertible, we can find the pseudoinverse of $\mathbf{\Sigma}$, denoted by $\mathbf{\Sigma}^\dagger$. In this case, we assume that $\tilde{\mathbf{\Phi}}$ is not invertible, therefore $\mathbf{\Sigma}$ is singular and pseudoinverse of it should be calculated. $\mathbf{\Sigma}^\dagger$ is found by taking the reciprocal of the diagomal elements, i.e., eigenvalues, and then taking the transpose of the matrix. Since $\mathbf{\Sigma}$ is singular, $\mathbf{\Sigma}^T\mathbf{\Sigma}$ is also non-invertible. Therefore, we find the pseudo-inverse of $\widetilde{\mathbf{K}}$, shown as $\widetilde{\mathbf{K}}^\dagger$, as

$$\widetilde{\mathbf{K}}^\dagger = \mathbf{V}\left(\mathbf{\Sigma}^T\mathbf{\Sigma}\right)^\dagger\mathbf{V}^T\,.$$

By multiplying both sides with $\tilde{\mathbf{\Phi}}$ from the left, we get

$$\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}^\dagger = \tilde{\mathbf{\Phi}}\mathbf{V}\left(\mathbf{\Sigma}^T\mathbf{\Sigma}\right)^\dagger\mathbf{V}^T\,. \tag{5.2.5}$$

When we substitute (5.2.1) into $\tilde{\mathbf{\Phi}}$ on the right-hand side of (5.2.5), we obtain

$$\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}^\dagger = \mathbf{U}\mathbf{\Sigma}\left(\mathbf{\Sigma}^T\mathbf{\Sigma}\right)^\dagger\mathbf{V}^T\,. \tag{5.2.6}$$

Note that (5.2.4) and (5.2.6) are equal to each other. Therefore, we obtain

$$\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}^\dagger = \frac{1}{n}\mathbf{C}^{-1}\tilde{\mathbf{\Phi}}\,. \tag{5.2.7}$$

We know that $\mathbf{C} = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T$ from (5.2.2). Therefore, using (5.1.3) $\mathbf{C}\,\tilde{\phi}(\mathbf{x})$ can be written as

$$\mathbf{C}\,\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T\left(\phi(\mathbf{x}) - \frac{1}{n}\mathbf{\Phi}\mathbf{1}_n\right). \tag{5.2.8}$$

By substituting the transpose of (5.1.6) into $\tilde{\mathbf{\Phi}}^T$ in (5.2.8), we get

$$\mathbf{C}\,\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\mathbf{H}\mathbf{\Phi}^T\left(\phi(\mathbf{x}) - \frac{1}{n}\mathbf{\Phi}\mathbf{1}_n\right) = \frac{1}{n}\tilde{\mathbf{\Phi}}\mathbf{H}\left(\mathbf{\Phi}^T\phi(\mathbf{x}) - \frac{1}{n}\mathbf{\Phi}^T\mathbf{\Phi}\mathbf{1}_n\right). \tag{5.2.9}$$

Note that $\mathbf{\Phi}^T\phi(\mathbf{x})$ is equal to $\bar{\mathbf{k}}_x$ and $\mathbf{\Phi}^T\mathbf{\Phi}$ is $\mathbf{K}$. Then (5.2.9) will be

$$\mathbf{C}\,\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\mathbf{H}\left(\bar{\mathbf{k}}_{\mathbf{x}} - \frac{1}{n}\mathbf{K}\mathbf{1}_n\right),$$

and using (5.1.14), it leads to

$$\mathbf{C}\,\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{k}}_{\mathbf{x}}\,. \tag{5.2.10}$$

We know that C is invertible. Using (5.2.10), we obtain

$$\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\mathbf{C}^{-1}\tilde{\mathbf{\Phi}}\tilde{\mathbf{k}}_{\mathbf{x}}. \tag{5.2.11}$$

Substitute (5.2.7) into (5.2.11) and get

$$\tilde{\phi}(\mathbf{x}) = \tilde{\mathbf{\Phi}}\,\widetilde{\mathbf{K}}^{\dagger}\,\tilde{\mathbf{k}}_{\mathbf{x}}\,. \tag{5.2.12}$$

Therefore, kernelized Mahalanobis distance for invertible covariance becomes

$$d^2_{IC}(\mathbf{x}) = d^2_{IC}(\phi(\mathbf{x}); \{\boldsymbol{\phi}_\mu, \mathbf{C}\}) = \tilde{\phi}(\mathbf{x})^T\mathbf{C}^{-1}\tilde{\phi}(\mathbf{x}) = \tilde{\phi}(\mathbf{x})^T\mathbf{C}^{-1}\tilde{\mathbf{\Phi}}\,\widetilde{\mathbf{K}}^{\dagger}\,\tilde{\mathbf{k}}_{\mathbf{x}}. \tag{5.2.13}$$

Note that when $\mathbf{C}^{-1}\tilde{\mathbf{\Phi}}$ is obtained from (5.2.7) and substituted into (5.2.13), the distance function becomes

$$d^2_{IC}(\mathbf{x}) = d^2_{IC}(\phi(\mathbf{x}); \{\boldsymbol{\phi}_\mu, \mathbf{C}\}) = n\,\tilde{\phi}(\mathbf{x})^T\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}^{\dagger}\widetilde{\mathbf{K}}^{\dagger}\tilde{\mathbf{k}}_{\mathbf{x}}$$

$$= \tilde{\phi}(\mathbf{x})^T\mathbf{C}^{-1}\tilde{\phi}(\mathbf{x}) = n\,\tilde{\mathbf{k}}_{\mathbf{x}}^T\left(\widetilde{\mathbf{K}}^{\dagger}\right)^2\tilde{\mathbf{k}}_{\mathbf{x}}\,. \tag{5.2.14}$$

5.2.1 Mahalanobis Distance between Two Data Points

Please note that (5.2.14) refers to the distance between $\phi(\tilde{\mathbf{x}})$ and the mean of the data set. However, we can find the distance between two data points which are centralized in *kernel space*, say $\tilde{\phi}(\mathbf{x})$ and $\tilde{\phi}(\mathbf{y})$. Mahalanobis distance function can be written as

$$\begin{aligned} d^2_{IC}(\mathbf{x},\mathbf{y}) &= d^2_{IC}(\phi(\mathbf{x}),\phi(\mathbf{y}); \{\boldsymbol{\phi}_\mu, \mathbf{C}\}) \\ &= [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\,\mathbf{C}^{-1}\,[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]\,. \end{aligned} \tag{5.2.15}$$

Following (5.2.12) the difference between $\tilde{\phi}(\mathbf{x})$ and $\tilde{\phi}(\mathbf{y})$ is

$$\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y}) = \tilde{\mathbf{\Phi}}\,\widetilde{\mathbf{K}}^{\dagger}[\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}]\,. \tag{5.2.16}$$

When (5.2.16) is substituted into (5.2.15), Mahalanobis distance becomes

$$d_{IC}^2(\mathbf{x},\mathbf{y}) = [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T \, \mathbf{C}^{-1} \, \tilde{\mathbf{\Phi}} \, \widetilde{\mathbf{K}}^{\dagger} [\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] \, . \tag{5.2.17}$$

For $\mathbf{C}^{-1} \, \tilde{\mathbf{\Phi}}$ in (5.2.17), (5.2.7) is substituted and we get

$$d_{IC}^2(\mathbf{x},\mathbf{y}) = n \, [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T \, \tilde{\mathbf{\Phi}} \, (\widetilde{\mathbf{K}}^{\dagger})^2 [\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] \, .$$

Following (5.1.10), $\tilde{\phi}(\mathbf{x})^T \, \tilde{\mathbf{\Phi}}$ and $\tilde{\phi}(\mathbf{y})^T \, \tilde{\mathbf{\Phi}}$ are equal to $\tilde{\mathbf{k}}_{\mathbf{x}}^T$ and $\tilde{\mathbf{k}}_{\mathbf{y}}^T$, respectively. Therefore, we obtain the Mahalanobis distance between the centralized data points $\mathbf{x}$ and $\mathbf{y}$ in the *kernel space* as

$$d_{IC}^2(\mathbf{x},\mathbf{y}) = d_{IC}^2(\tilde{\phi}(\mathbf{x}), \tilde{\phi}(\mathbf{y}); \{\boldsymbol{\phi}_{\mu}, \mathbf{C}\}) = n \, [\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}]^T \, (\widetilde{\mathbf{K}}^{\dagger})^2 [\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] \, .$$

5.3 Kernel Mahalanobis Distance for Regularized Covariance

When the dimension of $\mathscr{H}$ is higher than n or infinite, the covariance operator is non-invertible. Therefore, it is regularized so that it will not be singular. Regularized covariance, denoted by $\mathbf{C}_{reg}$, becomes

$$\mathbf{C}_{reg} = \mathbf{C} + \sigma_r^2 \mathbf{I}_{\mathscr{H}} = \frac{1}{n} \tilde{\mathbf{\Phi}} \tilde{\mathbf{\Phi}}^T + \sigma_r^2 \mathbf{I}_{\mathscr{H}} \, , \tag{5.3.1}$$

where $\mathbf{I}_{\mathscr{H}}$ is an identity matrix with Hilbert space dimension and σ_r is a predefined parameter. When (5.3.1) is multiplied with $\tilde{\mathbf{\Phi}}$ from the right, we obtain

$$\mathbf{C}_{reg} \tilde{\mathbf{\Phi}} = \frac{1}{n} \tilde{\mathbf{\Phi}} \tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}} + \sigma_r^2 \mathbf{I}_{\mathscr{H}} \tilde{\mathbf{\Phi}} \, . \tag{5.3.2}$$

Note that $\tilde{\mathbf{\Phi}}^T \tilde{\mathbf{\Phi}}$ is equal to $\widetilde{\mathbf{K}}$. Therefore, (5.3.2) can be written as

$$\mathbf{C}_{reg} \tilde{\mathbf{\Phi}} = \frac{1}{n} \tilde{\mathbf{\Phi}} \left(\widetilde{\mathbf{K}} + n\sigma_r^2 \mathbf{I}_n \right) ,$$

which can be defined as

$$\mathbf{C}_{reg} \tilde{\mathbf{\Phi}} = \frac{1}{n} \tilde{\mathbf{\Phi}} \widetilde{\mathbf{K}}_{reg} \, , \tag{5.3.3}$$

where $\widetilde{\mathbf{K}}_{reg} = \widetilde{\mathbf{K}} + n\sigma_r^2 \mathbf{I}_n$. When $n\sigma_r^2 > 0$, then $\mathbf{C}_{reg}$ and $\widetilde{\mathbf{K}}_{reg}$ become strictly positive definite and nonsingular. Multiplying (5.3.3) with $\mathbf{C}_{reg}$ from the left and $\widetilde{\mathbf{K}}_{reg}$ from the right gives

$$\tilde{\mathbf{\Phi}} \widetilde{\mathbf{K}}_{reg}^{-1} = \frac{1}{n} \mathbf{C}_{reg}^{-1} \tilde{\mathbf{\Phi}} \, . \tag{5.3.4}$$

When $\mathbf{C}_{reg}$ in (5.3.1) is multiplied with $\tilde{\phi}(\mathbf{x})$, we obtain

$$\mathbf{C}_{reg}\,\tilde{\phi}(\mathbf{x}) = \left(\frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T + \sigma_r^2\mathbf{I}_{\mathscr{H}}\right)\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{\Phi}}^T\tilde{\phi}(\mathbf{x}) + \sigma_r^2\mathbf{I}_{\mathscr{H}}\tilde{\phi}(\mathbf{x})\,. \tag{5.3.5}$$

From $\tilde{\mathbf{\Phi}}^T\tilde{\phi}(\mathbf{x}) = \tilde{\mathbf{k}}$, (5.3.5) will be

$$\mathbf{C}_{reg}\,\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\mathbf{\Phi}}\tilde{\mathbf{k}}_{\mathbf{x}} + \sigma_r^2\tilde{\phi}(\mathbf{x})\,. \tag{5.3.6}$$

Since $\mathbf{C}_{reg}$ is invertible, multiply (5.3.6) with $\mathbf{C}_{reg}^{-1}$ from the left. Then

$$\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\mathbf{C}_{reg}^{-1}\tilde{\mathbf{\Phi}}\tilde{\mathbf{k}}_{\mathbf{x}} + \sigma_r^2\mathbf{C}_{reg}^{-1}\tilde{\phi}(\mathbf{x})\,. \tag{5.3.7}$$

Afterwards, multiplying each side of (5.3.7) with $\tilde{\phi}(\mathbf{x})^T$ from the left gives

$$\tilde{\phi}(\mathbf{x})^T\tilde{\phi}(\mathbf{x}) = \frac{1}{n}\tilde{\phi}(\mathbf{x})^T\mathbf{C}_{reg}^{-1}\tilde{\mathbf{\Phi}}\tilde{\mathbf{k}}_{\mathbf{x}} + \sigma_r^2\tilde{\phi}(\mathbf{x})^T\mathbf{C}_{reg}^{-1}\tilde{\phi}(\mathbf{x})\,. \tag{5.3.8}$$

Substitute (5.3.4) into (5.3.8), we get

$$\tilde{\phi}(\mathbf{x})^T\tilde{\phi}(\mathbf{x}) = \tilde{\phi}(\mathbf{x})^T\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}_{reg}^{-1}\tilde{\mathbf{k}}_{\mathbf{x}} + \sigma_r^2\tilde{\phi}(\mathbf{x})^T\mathbf{C}_{reg}^{-1}\tilde{\phi}(\mathbf{x})\,. \tag{5.3.9}$$

Then the second term, $\tilde{\phi}(\mathbf{x})^T\mathbf{C}_{reg}^{-1}\tilde{\phi}(\mathbf{x})$, in (5.3.9) will be

$$\tilde{\phi}(\mathbf{x})^T\mathbf{C}_{reg}^{-1}\tilde{\phi}(\mathbf{x}) = \frac{\tilde{\phi}(\mathbf{x})^T\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{x})^T\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}_{reg}^{-1}\tilde{\mathbf{k}}_{\mathbf{x}}}{\sigma_r^2}.$$

which is equal to kernel Mahalanobis distance for regularized covariance, shown by $d_{RC}^2(\mathbf{x})$. From $\tilde{\phi}(\mathbf{x})^T\tilde{\phi}(\mathbf{x}) = \tilde{k}_{\mathbf{xx}}$ and $\tilde{\phi}(\mathbf{x})^T\tilde{\mathbf{\Phi}} = \tilde{\mathbf{k}}_{\mathbf{x}}^T$,

$$d_{RC}^2(\mathbf{x}) = d^2(\phi(\mathbf{x}); \{\boldsymbol{\phi}_\mu, \mathbf{C}_{reg}\}) = \frac{1}{\sigma_r^2}\left(\tilde{k}_{\mathbf{xx}} - \tilde{\mathbf{k}}_{\mathbf{x}}^T\mathbf{K}_{reg}^{-1}\tilde{\mathbf{k}}_{\mathbf{x}}\right)\,. \tag{5.3.10}$$

5.3.1 Mahalanobis Distance between Two Data Points

As in (5.2.14), the expression (5.3.10) provides the distance between $\phi(\mathbf{x})$ and the mean of the centralized data points in the *kernel space*. Following (5.3.10), we can also find the distance between two centralized data points in the kernel space. Using the equation in (5.3.7), the difference between two data points will be

$$\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y}) = \frac{1}{n}\mathbf{C}_{reg}^{-1}\tilde{\mathbf{\Phi}}[\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] + \sigma_r^2\mathbf{C}_{reg}^{-1}[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]\,. \tag{5.3.11}$$

If we multiply both sides with $[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T$, it becomes

$$[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})] = \frac{1}{n}[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\mathbf{C}_{reg}^{-1}\tilde{\mathbf{\Phi}}[\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] + \sigma_r^2[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\mathbf{C}_{reg}^{-1}[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]\,. \tag{5.3.12}$$

When (5.3.4) is substituted for $\frac{1}{n}\mathbf{C}_{reg}^{-1}\tilde{\mathbf{\Phi}}$ in (5.3.12), we obtain

$$[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})] = [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}_{reg}^{-1}[\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}] + \sigma_r^2[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\mathbf{C}_{reg}^{-1}[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]\,.$$

Then the Mahalanobis distance between $\phi(\tilde{\mathbf{x}})$ and $\phi(\tilde{\mathbf{y}})$ is

$$d_{RC}^2(\mathbf{x},\mathbf{y}) = d_{RC}^2(\phi(\tilde{\mathbf{x}}),\phi(\tilde{\mathbf{y}});\{\boldsymbol{\phi}_\mu,\mathbf{C}\}) = [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\mathbf{C}_{reg}^{-1}[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]$$

$$= \frac{1}{\sigma_r^2}\left([\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T[\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})] - [\tilde{\phi}(\mathbf{x}) - \tilde{\phi}(\mathbf{y})]^T\tilde{\mathbf{\Phi}}\widetilde{\mathbf{K}}_{reg}^{-1}[\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}}]\right)\,. \tag{5.3.13}$$

Again from $\tilde{\phi}(\mathbf{x})^T\,\tilde{\mathbf{\Phi}} = \tilde{\mathbf{k}}_{\mathbf{x}}^T$ and $\tilde{\phi}(\mathbf{y})^T\,\tilde{\mathbf{\Phi}} = \tilde{\mathbf{k}}_{\mathbf{y}}^T$, (5.3.13) will be

$$d_{RC}^2(\mathbf{x},\mathbf{y}) = \frac{1}{\sigma_r^2}\left[(\tilde{k}_{\mathbf{xx}} - 2\tilde{k}_{\mathbf{xy}} + \tilde{k}_{\mathbf{yy}}) - (\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}})^T\widetilde{\mathbf{K}}_{reg}^{-1}(\tilde{\mathbf{k}}_{\mathbf{x}} - \tilde{\mathbf{k}}_{\mathbf{y}})\right]\,.$$

Note that $\tilde{k}_{\mathbf{xy}}$ is the inner product of mapped $\mathbf{x}$ and mapped $\mathbf{y}$.

5.4 Kernel Pd-clustering with Mahalanobis Distance

Kernel Pd-clustering with Mahalanobis distance algorithm (KPD-M) uses Mahalanobis distance as a distance measure. The Mahalanobis distance between two data points is shown as

$$\|\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\|_M = \sqrt{\Big(\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\Big)^T\mathbf{C}^{-1}\Big(\phi(\mathbf{x}_i) - \phi(\mathbf{x}_j)\Big)}\,.$$

It defines and updates the cluster centers in the *kernel space* as in **KPD Algorithm**. The cluster centers cannot be calculated explicitly; therefore, they will be written as a linear combination of data points. Using (4.3.8), the optimization problem becomes

$$\min \qquad \sum_{i=1}^{N}\sum_{j=1}^{T} p_{ij}^2\,\|\phi(\mathbf{x}_i) - \sum_{i=1}^{N}\beta_{ij}\,\phi(\mathbf{x}_i)\|_M \tag{5.4.1}$$

$$\text{s.t.} \qquad \sum_{j=1}^{T} p_{ij} = 1 \qquad \forall i$$

$$p_{ij} \geq 0, \qquad \forall i, j$$

where

$$\|\phi\left(\mathbf{x}_i\right) - \sum_{i=1}^{N} \beta_{ij}\, \phi(\mathbf{x}_i)\|_M = \sqrt{\left(\phi\left(\mathbf{x}_i\right) - \sum_{i=1}^{N} \beta_{ij}\, \phi(\mathbf{x}_i)\right)^T \mathbf{C}^{-1} \left(\phi\left(\mathbf{x}_i\right) - \sum_{i=1}^{N} \beta_{ij}\, \phi(\mathbf{x}_i)\right)}\,.$$

The Mahalanobis distance above will be calculated using kernel Mahalanobis distance with regularized covariance as in Section 5.3 . Rearranging (5.3.11) gives

$$\begin{aligned} \tilde{\phi}(\mathbf{x}_i) - \sum_{i=1}^{N} \beta_{ij} \tilde{\phi}(\mathbf{x}_i) = \frac{1}{n} \mathbf{C}_{reg}^{-1} \tilde{\mathbf{\Phi}} \left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N} \beta_{ij} \tilde{\mathbf{k}}_{\mathbf{x}_i}\right) \\ + \sigma_r^2 \mathbf{C}_{reg}^{-1} \left(\tilde{\phi}(\mathbf{x}_i) - \sum_{i=1}^{N} \beta_{ij} \tilde{\phi}(\mathbf{x}_i)\right). \end{aligned} \tag{5.4.2}$$

Let $(\tilde{\phi}(\mathbf{x}_i) - \sum_{i=1}^{N} \beta_{ij} \tilde{\phi}(\mathbf{x}_i))$ be A_{ij}. Then multiplying each side in (5.4.2) with A_{ij}^T yields

$$A_{ij}^T A_{ij} = A_{ij}^T \frac{1}{n} \mathbf{C}_{reg}^{-1} \tilde{\mathbf{\Phi}} \left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N} \beta_{ij} \tilde{\mathbf{k}}_{\mathbf{x}_i}\right) + \sigma_r^2 A_{ij}^T \mathbf{C}_{reg}^{-1} A_{ij}\,. \tag{5.4.3}$$

When (5.3.4) is substituted for $\frac{1}{n} \mathbf{C}_{reg}^{-1} \tilde{\mathbf{\Phi}}$ in (5.4.3), we obtain

$$A_{ij}^T A_{ij} = A_{ij}^T \tilde{\mathbf{\Phi}} \widetilde{K}_{reg}^{-1} \left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N} \beta_{ij} \tilde{\mathbf{k}}_{\mathbf{x}_i}\right) + \sigma_r^2 A_{ij}^T \mathbf{C}_{reg}^{-1} A_{ij}\,. \tag{5.4.4}$$

Note that $A_{ij}^T \mathbf{C}_{reg}^{-1} A_{ij}$ in (5.4.4) gives the squared Mahalanobis distance between $\tilde{\phi}(\mathbf{x}_i)$ and $\sum_{i=1}^{N} \beta_{ij} \tilde{\phi}(\mathbf{x}_i)$. Then the distance is found as

$$A_{ij}^T \mathbf{C}_{reg}^{-1} A_{ij} = \frac{1}{\sigma_r^2} \left(A_{ij}^T A_{ij} - A_{ij}^T \tilde{\mathbf{\Phi}} \widetilde{K}_{reg}^{-1} \left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N} \beta_{ij} \tilde{\mathbf{k}}_{\mathbf{x}_i}\right) \right).$$

From $\tilde{\phi}(\mathbf{x}_i)^T\,\tilde{\mathbf{\Phi}} = \tilde{\mathbf{k}}_{\mathbf{x}_i}^T$, $A_{ij}^T\tilde{\mathbf{\Phi}}$ becomes $\left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N}\beta_{ij}\tilde{\mathbf{k}}_{\mathbf{x}_i}\right)^T$, and

$$\begin{aligned}
&\|\tilde{\phi}\,(\mathbf{x}_i) - \sum_{k=1}^{N}\beta_{kt}\,\tilde{\phi}(\mathbf{x}_k)\|_M^2 \\
&= \frac{1}{\sigma_r^2}\left[\left(\tilde{k}_{\mathbf{x}_i\mathbf{x}_i} - 2\sum_{l=1}^{N}\beta_{lj}\tilde{k}_{\mathbf{x}_i\mathbf{x}_l} + \sum_{l=1}^{N}\sum_{h=1}^{N}\beta_{lj}\beta_{hj}\tilde{k}_{\mathbf{x}_l\mathbf{x}_h}\right)\right. \\
&\left.-\left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N}\beta_{ij}\tilde{\mathbf{k}}_{\mathbf{x}_i}\right)^T\widetilde{\mathbf{K}}_{reg}^{-1}\left(\tilde{\mathbf{k}}_{\mathbf{x}_i} - \sum_{i=1}^{N}\beta_{ij}\tilde{\mathbf{k}}_{\mathbf{x}_i}\right)\right]\,.
\end{aligned} \tag{5.4.5}$$

The distance in (5.4.5) considers the centered data points. Coefficient and probability calculations in (5.4.12) and (5.4.13) use the distance between non-centered data points. (5.4.5) can be directly used in those calculations because centering of data points will not change the distance value. Since we need the Mahalanobis distance itself, $\|\phi\,(\mathbf{x}_i) - \sum_{k=1}^{N}\beta_{kj}\,\phi(\mathbf{x}_k)\|_M$ is equal to the square root of (5.4.5).

The Mahalanobis distance $\|\phi\,(\mathbf{x}_i) - \sum_{k=1}^{N}\beta_{kj}\,\phi(\mathbf{x}_k)\|_M$ can be written in terms of matrices and vectors. As in Section 4.3.1, let $\boldsymbol{\beta}_j$ be the column vector containing all β_{ij}'s for a given cluster j. $\widetilde{\mathbf{K}}$ is the kernel matrix of centered data points. Thus, $\widetilde{\mathbf{K}}_i$ becomes the column of that matrix and $\widetilde{\mathbf{K}}_{ii}$ is the i^{th} row and i^{th} column element. Representation of the Mahalanobis distance in terms of $\boldsymbol{\beta}_j$ and $\widetilde{\mathbf{K}}$ becomes

$$\sqrt{\frac{1}{\sigma_r^2}\left[\widetilde{\mathbf{K}}_{ii} - 2\,\boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}_i + \boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j - \left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)^T\widetilde{\mathbf{K}}_{reg}^{-1}\left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)\right]}\,. \tag{5.4.6}$$

Using the Mahalanobis distance obtained in (5.4.6), the objective function (5.4.1) can be written as

$$\sum_{i=1}^{N}\sum_{j=1}^{T}p_{ij}^2\sqrt{\frac{1}{\sigma_r^2}\left[\widetilde{\mathbf{K}}_{ii} - 2\,\boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}_i + \boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j - \left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)^T\widetilde{\mathbf{K}}_{reg}^{-1}\left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)\right]}\,.$$

Then the Lagrangian $L(\mathbf{P}, \mathbf{C}, \mathbf{\Lambda})$ becomes

$$\begin{aligned}
&\sum_{i=1}^{N}\sum_{j=1}^{T}p_{ij}^2\sqrt{\frac{1}{\sigma_r^2}\left[\widetilde{\mathbf{K}}_{ii} - 2\,\boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}_i + \boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j - \left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)^T\widetilde{\mathbf{K}}_{reg}^{-1}\left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)\right]} \\
&-\sum_{i=1}^{N}\lambda_i\left(\sum_{j=1}^{T}p_{ij} - 1\right).
\end{aligned} \tag{5.4.7}$$

The derivative of (5.4.7) with respect to $\boldsymbol{\beta}_j$ is found as

$$\frac{\partial L}{\partial \boldsymbol{\beta}_j} = \frac{1}{\sigma_r^2} \sum_{i=1}^{N} \frac{1}{2}\, p_{ij}^2\, \frac{-2\,\widetilde{\mathbf{K}}_i + 2\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j + \left(2\,\widetilde{\mathbf{K}}^T\,\mathbf{K}_{reg}^{-1}\big(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\big)\right)}{d_M(i,j)}\,, \tag{5.4.8}$$

where $d_M(i,j)$ is

$$\sqrt{\frac{1}{\sigma_r^2}\left[\widetilde{\mathbf{K}}_{ii} - 2\,\boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}_i + \boldsymbol{\beta}_j^T\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j - \left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)^T \widetilde{\mathbf{K}}_{reg}^{-1}\left(\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j\right)\right]}\,. \tag{5.4.9}$$

Making (5.4.8) equal to 0 yields

$$\sum_{i=1}^{N} p_{ij}^2\, \frac{\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j - \widetilde{\mathbf{K}}^T\,\mathbf{K}_{reg}^{-1}\,\widetilde{\mathbf{K}}\,\boldsymbol{\beta}_j}{d_M(i,j)} = \sum_{i=1}^{N} p_{ij}^2\, \frac{\widetilde{\mathbf{K}}_i - \widetilde{\mathbf{K}}^T\,\mathbf{K}_{reg}^{-1}\,\widetilde{\mathbf{K}}_i}{d_M(i,j)}\,.$$

By rearranging the above equation, we obtain

$$\left(\widetilde{\mathbf{K}} - \widetilde{\mathbf{K}}^T\,\mathbf{K}_{reg}^{-1}\,\widetilde{\mathbf{K}}\right)\boldsymbol{\beta}_j \sum_{i=1}^{N} \frac{p_{ij}^2}{d_M(i,j)} = \sum_{i=1}^{N} p_{ij}^2\, \frac{\left(\mathbf{I} - \widetilde{\mathbf{K}}^T\,\mathbf{K}_{reg}^{-1}\right)\widetilde{\mathbf{K}}_i}{d_M(i,j)}\,. \tag{5.4.10}$$

Assuming that $\widetilde{\mathbf{K}}$ matrix is invertible, multiplying both sides of (5.4.10) with $\widetilde{\mathbf{K}}^{-1}$ from the left gives

$$\left(\mathbf{I} - \mathbf{K}_{reg}^{-1}\,\widetilde{\mathbf{K}}\right)\boldsymbol{\beta}_j \sum_{i=1}^{N} \frac{p_{ij}^2}{d_M(i,j)} = \left(\widetilde{\mathbf{K}}^{-1} - \mathbf{K}_{reg}^{-1}\right) \sum_{i=1}^{N} p_{ij}^2\, \frac{\widetilde{\mathbf{K}}_i}{d_M(i,j)}\,. \tag{5.4.11}$$

Please note that since $\widetilde{\mathbf{K}}$ matrix is symmetric, $\widetilde{\mathbf{K}}^{-1}\,\widetilde{\mathbf{K}}^T = \mathbf{I}$. From (5.4.11), $\boldsymbol{\beta}_j$ is obtained as

$$\boldsymbol{\beta}_j = \left(\mathbf{I} - \mathbf{K}_{reg}^{-1}\,\widetilde{\mathbf{K}}\right)^{-1}\left(\widetilde{\mathbf{K}}^{-1} - \mathbf{K}_{reg}^{-1}\right) \frac{\displaystyle\sum_{i=1}^{N} p_{ij}^2\, \frac{\widetilde{\mathbf{K}}_i}{d_M(i,j)}}{\displaystyle\sum_{i=1}^{N} \frac{p_{ij}^2}{d_M(i,j)}}\,. \tag{5.4.12}$$

For given $\boldsymbol{\beta}_j$'s, p_{ij}'s are found as by taking the derivative of (5.4.7) with respect to p_{ij}'s. We obtain

$$p_{ij} = \frac{\dfrac{1}{d_M(i,j)}}{\displaystyle\sum_{t=1}^{T} \frac{1}{d_M(i,t)}}\,. \tag{5.4.13}$$

5.4.1 KPD-M Algorithm

KPD-M Algorithm updates the cluster centers implicitly as in **KPD Algorithm**. Therefore, it is enough to update $\boldsymbol{\beta}_j$'s and p_{ij}'s only. The update rule for $\boldsymbol{\beta}_j$'s becomes

$$\boldsymbol{\beta}_j^{(r)} = \left(\mathbf{I} - \mathbf{K}_{reg}^{-1}\widetilde{\mathbf{K}}\right)^{-1}\left(\widetilde{\mathbf{K}}^{-1} - \mathbf{K}_{reg}^{-1}\right)\frac{\sum_{i=1}^{N}\frac{(p_{ij}^{(r-1)})^2\,\widetilde{\mathbf{K}}_i}{d_M(i,j)^{(r-1)}}}{\sum_{i=1}^{N}\frac{(p_{ij}^{(r-1)})^2}{d_M(i,j)^{(r-1)}}}\cdot \tag{5.4.14}$$

After all $\boldsymbol{\beta}_j$'s are updated, using new $\boldsymbol{\beta}_j$ values p_{ij}'s will be updated as

$$p_{ij}^{(r)} = \frac{\frac{1}{d_M(i,j)^{(r)}}}{\sum_{t=1}^{T}\frac{1}{d_M(i,t)^{(r)}}}\cdot \tag{5.4.15}$$

The pseudocode of **KPD-M Algorithm** is given in Algorithm 4.

Algorithm 4: KPD-M Algorithm

Input : data set $\mathbf{X}$, coefficient matrix $\mathbf{B}$, number of clusters T, kernel function k, stopping criterion ϵ, regularization parameter σ_r

Output: probability matrix $\mathbf{P}$

Initialize $\mathbf{P}$ to a random probability matrix and $\mathbf{B}$ to a random coefficient matrix and Calculate $K(\mathbf{x}_i, \mathbf{x}_l)$ and $\tilde{k}_{\mathbf{x}_i\mathbf{x}_l}$ for $i, l \in 1, ..., N$, $\tilde{\mathbf{k}}_{\mathbf{x}_i}$ for $i \in 1, ..., N$, and $\widetilde{\mathbf{K}}_{reg}^{-1}$.

Calculate $p_{ij}^{(0)}$'s in $\mathbf{P}$ matrix as in (5.4.15).

Set $r = 0$.

while $\sum_{j=1}^{T} |\boldsymbol{\beta}_j^{(r)} - \boldsymbol{\beta}_j^{(r-1)}| > \epsilon$ **do**

  $r = r + 1$

  Update $\boldsymbol{\beta}_j^{(r)}$ as in (5.4.14)

  Update $p_{ij}^{(r)}$ as in (5.4.15)

end

CHAPTER 6

EXPERIMENTAL STUDY

In this section, experimental study for the proposed algorithms KPD, KPD-F, and KPD-M are conducted. These algorithms are coded in MATLAB R2021b, and they are run using a computer with Intel Xeon E2246G, having a 3.6 GHz processor, and 16 GB RAM. For each algorithm, real data sets from UCI machine learning repository and synthetic data sets are used for the experiments, and performance of them are measured. The experiment results of the proposed algorithms are compared with the best known results coming from soft kernel methods for each data set. These results are taken from the study [29] with the algorithms of KFCM-F and KFCM-K.

Eight real data sets are chosen from UCI machine learning data sets [37], namely Iris, Wisconsin Diagnostic Breast Cancer, Wine, Wisconsin (Original) Breast Cancer [38], Ionosphere, Haberman, Sonar, and SPECT data sets. For the synthetic data sets, different patterns of data are generated, and they are named as Ring, Line, Dense, Fuzzy X, Parabolic, and Noisy Ring data sets. All data sets are normalized to have zero mean and one standard deviation.

Polynomial (P) and Gaussian (G) kernel functions are used for the experimentation. For the kernel functions, several parameter values are explored. Table 6.1 shows the parameter settings for the kernel functions.

Table 6.1: Parameter settings for kernel functions

Kernel Function	Parameter	Value
Polynomial	a	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 20, 30, 40, 50
	b	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
Gaussian	σ	0.01, 0.02, 0.05, 0.1-10*, 20, 30, 40, 50, 60, 70, 80, 90, 100

* Values are incremented by 0.1 starting from 0.1 until 10.

Here, polynomial kernel has two parameters to be determined, constant a and exponent b. For the constant a, 17 different values ranging between 0 and 50 are tested. The values for the exponent term b vary between 2 and 12. Therefore, 17×11 different parameter setting is experimented for each algorithm that use polynomial kernel to determine the kernel parameters.

Gaussian kernel has only one parameter σ. For the experimentation, 112 different σ values between 0.01 and 100 are tested. For each parameter value, algorithms are run 100 times because of the randomness in the algorithms, and the parameters giving the best result on the average accuracy are chosen for each algorithm.

For KPD-M algorithms, σ_r parameter used in the distance function (5.4.9) should also be determined. Therefore, 10 different σ_r values, shown in Table 6.2, are tested for both KPD-M algorithms.

Table 6.2: σ_r values for KPD-M algorithms

Parameter	Values
σ_r	0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20

Among those parametres, the value that gives the best average accuracy level is set as σ_r.

KPD and KPD-M algorithms have a stopping criterion of

$$\sum_{j=1}^{T} |\boldsymbol{\beta}_j^{(r)} - \boldsymbol{\beta}_j^{(r-1)}| < 10^{-10} .$$

That is, when the sum of the change of all the cluster coefficients becomes less than

10^{-10}, the algorithm stops. For KPD-F algorithms, the stopping criterion is determined as

$$\sum_{j=1}^{T} |\mathbf{c}_j^{(r)} - \mathbf{c}_j^{(r-1)}| < 10^{-10}\,,$$

which means that the algorithm will stop when the sum of the change in the centers in the input space is less than 10^{-10}. In all of the algorithms, we also add another stopping criterion that ensures that the total number of iterations does not exceed 100 in order to prevent looping.

We report the results of the algorithms for both polynomial (P) and Gaussian (G) kernels. Since we propose three algorithms and use two kernel functions for each algorithm, six algorithm results are provided for each data set. Note that, [29] reports KFCM-F algorithm results for Gaussian kernel only, while giving KFCM-K results for both of the kernel functions.

KPD-M algorithm is developed using the Mahalanobis distance with regularized covariance only. Mahalanobis distance with invertible covariance in Section 5.2 is not taken into account since it is not known whether the covariance matrix is invertible.

In the result tables provided for each data set, the first column gives the algorithms to be tested with their optimal parameters. The second, third, and fourth columns provide the best, worst, and average accuracy levels, respectively. Best accuracy is obtained by taking the maximum accuracy level out of 100 trials. The minimum accuracy of 100 trials is reported as worst accuracy. Average accuracy level is found by taking the mean of the accuracy of 100 trials. Fifth column provides the average CPU time of the algorithms. The number of times the best accuracy level is obtained in these trials, called best hit, is supplied in the last column.

Kernel probabilistic distance clustering algorithms are soft clustering methods. Therefore, instead of assigning a data point to a single cluster, they calculate the probability of a data point to belong to a cluster. To make a hard clustering assignment, we assign a data point to a cluster such that the probability of data point belonging to a cluster

is the highest. That is, given $\mathbf{x}_i$,

$$\mathbf{x}_i \in \mathbf{A}_j \text{ if } j = \arg\max_j \{p_{ij}\}\,,$$

where $\mathbf{A}_j$ is the set of data points in cluster j.

Label of a cluster is determined by choosing the majority class of data points in that cluster. After cluster labels are specified, the original class of data points and their assigned class coming from clusters they belong are compared. The accuracy is obtained by dividing the summation of data points with correct matching into total number of data points. That is,

$$\text{accuracy} = \frac{\sum_{i=1}^{N} \delta_{y_i,\hat{y}_i}}{N},$$

where N is the number of objects, y_i is the class of object i, $\hat{y}_i$ is the assigned class of object i (or the class of cluster that object i belongs to), and $\delta_{y_i,\hat{y}_i}$ is the Kronecker delta function such that

$$\delta_{y_i,\hat{y}_i} = \begin{cases} 1, & \text{if } y_i = \hat{y}_i\,, \\ 0, & \text{if } y_i \neq \hat{y}_i\,. \end{cases}$$

6.1 Real Data Sets

UCI Machine learning data sets that are used are multivariate data sets, and each data point belongs to a class. According to those labels, clustering accuracy is calculated by checking whether data points having the same label are clustered together.

6.1.1 Iris Data Set

Iris data set consists of three classes and four attributes. Data set contains 150 objects, and there are no missing values. Each class has the equal number of data points. Table 6.3 shows the clustering algorithm results of the proposed algorithms and kernel algorithms in the literature for the Iris data with three clusters. The optimal parameter values for each algorithm are also supplied.

When the best results are examined, KPD (G) and KPD-M (G) can construct clusters with 100% accuracy for 11 and 8 times out of 100 trials, respectively. Variance of the accuracy level of these algorithms are high, and they have low average accuracy. When the average results are investigated, KPD-F (P) has the highest accuracy. It finds 96.7% accuracy in 99 trials. On the other hand, KPD-F (G) algorithm has a high average accuracy with no deviation. When proposed algorithms are compared with the ones in the literature, it can be seen that KPD-F (P) and KPD-F (G) performs very well on the average. KPD (G) and KPD-M (G) can obtain perfect clustering.

While KPD (P) has the highest CPU time, KPD-F (G) has the lowest one. However, considering the fact that the CPU time unit is seconds, CPU time of the algorithms are close to each other, and there are no significant difference between them.

Table 6.3: Results for Iris data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 10$)	84.0%	84.0%	84.0%	-	-
KFCM-K (G) ($\sigma = 2$)	85.3%	85.3%	85.3%	-	-
KFCM-K (P) ($a = 15, b = 8$)	88.7%	88.7%	88.7%	-	-
KPD (P) ($a = 2, b = 3$)	99.3%	66.0%	**87.4**%	0.767	11
KPD (G) ($\sigma = 0.78$)	**100.0**%	66.0%	75.0%	0.413	11
KPD-F (P) ($a = 10, b = 2$)	98.7%	56.0%	**97.2**%	0.015	99
KPD-F (G) ($\sigma = 7.8$)	96.7%	96.7%	**96.7**%	0.009	100
KPD-M (P) ($a = 9, b = 4$, $\sigma_r = 10$)	97.3%	54.0%	78.1%	0.555	10
KPD-M (G) ($\sigma = 0.8$, $\sigma_r = 2$)	**100.0**%	63.3%	77.0%	0.483	8

6.1.2 Wisconsin Diagnostic Breast Cancer Data Set

Wisconsin Diagnostic Breast Cancer data consists of 569 data points, each having 30 attributes with no missing values. There are two classes that define the data set. Therefore, in the experiments the cluster number is set to two.

Table 6.4 indicates that when the results are examined in terms of the average accu-

racy, KFCM-F (G) gives the highest accuracy, which is 92.8%. KPD-M (G) provides an average accuracy of 91.5%, which is the highest accuracy compared to the results of our proposed algorithms. The best accuracy is obtained by KPD-F (P) algorithm, and this result is observed 31 times in 100 trials. KPD-M (G), KPD-M (P), and KPD (P) algorithms also provide high best accuracy results. The CPU time of KPD-F methods are the lowest while the highest time is observed in KPD (G) algorithm.

Table 6.4: Results for Wisconsin diagnostic breast cancer data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 6.32$)	92.8%	92.8%	92.8%	-	-
KFCM-K (G) ($\sigma = 10$)	91.6%	91.6%	91.6%	-	-
KFCM-K (P) ($a = 50, b = 2$)	91.2%	91.2%	91.2%	-	-
KPD (P) ($a = 30, b = 2$)	91.9%	70.1%	86.0%	12.419	3
KPD (G) ($\sigma = 2$)	88.1%	80.6%	84.9%	24.309	3
KPD-F (P) ($a = 2, b = 7$)	**95.3**%	77.3%	85.1%	0.257	31
KPD-F (G) ($\sigma = 9.8$)	86.3%	84.7%	84.7%	0.109	1
KPD-M (P) ($a = 12, b = 2, \sigma_r = 10$)	**93.3**%	61.8%	81.8%	18.617	2
KPD-M (G) ($\sigma = 9.1, \sigma_r = 5$)	**94.9**%	88.1%	**91.5**%	15.285	6

6.1.3 Wine Data Set

Wine data has 178 instances, each has 13 attributes. The data set consists of three different classes, and classes have 59, 71, and 48 objects. There is no missing value. The experiment is conducted for three clusters. Table 6.5 shows the results.

The average accuracy level of the proposed algorithm ranges between 75.4% to 78.5%. On the other hand, the kernel algorithms in the literature yields 96-98% accuracy with no deviation. Therefore, our algorithms does not provide good results when the average accuracy is considered. When the best accuracy values are examined, KPD (G) and KPD-M (G) give 98.3 % and 98.9 % accuracy, respectively. They obtain the best result two times since these algorithms have higher variance. KPD-F (G) algorithm has the lowest variance, and gives highest average accuracy when only proposed al-

gorithms are considered. CPU time of the algorithms are so close to each other, and KPD-F algorithms have the minimum value.

Table 6.5: Results for Wine data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 1.41$)	96.1%	96.1%	96.1%	-	-
KFCM-K (G) ($\sigma = 3.46$)	97.8%	97.8%	97.8%	-	-
KFCM-K (P) ($a = 20, b = 2$)	97.8%	97.8%	97.8%	-	-
KPD (P) ($a = 3, b = 2$)	97.2%	62.3%	77.3%	0.565	5
KPD (G) ($\sigma = 1.4$)	**98.3**%	63.3%	75.6%	0.401	2
KPD-F (P) ($a = 2, b = 6$)	84.8%	75.3%	77.4%	0.078	4
KPD-F (G) ($\sigma = 9.8$)	78.7%	73.0%	78.5%	0.041	93
KPD-M (P) ($a = 2, b = 2$, $\sigma_r = 10$)	97.2%	62.2%	76.9%	0.644	2
KPD-M (G) ($\sigma = 1.3$, $\sigma_r = 1$)	**98.9**%	61.7%	75.4%	0.619	2

6.1.4 Wisconsin Original Breast Cancer Data Set

This data set contains 699 data points with nine attributes. 16 data points have missing values, therefore those objects were excluded in the experiments. The data set has two classes with a size of 444 and 239 (after objects with missing values are eliminated). Table 6.6 shows the experiment results.

KPD-M (P), KPD-F (P), and KPD-F (G) algorithms yield good results on Wisconsin original data set, and their results are better than other kernel algorithms reported. KPD-M (P) algorithm has the highest average accuracy rate when compared with all of the algorithms. It finds the best result 92 times. The worst result it gives is 96.3%. KPD-F (P) and KPD-F (G) algorithms find the second highest average accuracy rate. These algorithms have zero variance, and they have the smallest CPU time. KPD (G) provides 96.6% accuracy with no deviation, and beats KFCM-K (P) algorithm. However, it is costly in terms of CPU time. KPD (P) has the highest best accuracy level, which is observed 62 times in 100 trials. It has a higher average accuracy level although it has a deviation. KPD-M (G) is the only algorithm that does not find a

good result. Its clustering accuracy is the lowest compared to all the algorithms.

Table 6.6: Results for Wisconsin (Original) breast cancer data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 1.41$)	97.1%	97.1%	97.1%	-	-
KFCM-K (G) ($\sigma = 6.32$)	97.1%	97.1%	97.1%	-	-
KFCM-K (P) ($a = 50, b = 2$)	95.2%	95.2%	95.2%	-	-
KPD (P) ($a = 40, b = 7$)	**97.7**%	74.5%	96.4%	12.894	62
KPD (G) ($\sigma = 1.7$)	96.6%	96.6%	96.6%	51.184	100
KPD-F (P) ($a = 50, b = 2$)	**97.2**%	97.2%	**97.2**%	0.050	100
KPD-F (G) ($\sigma = 20$)	**97.2**%	97.2%	**97.2**%	0.023	100
KPD-M (P) ($a = 12, b = 2, \sigma_r = 10$)	**97.5**%	96.3%	**97.4**%	27.965	92
KPD-M (G) ($\sigma = 7.7, \sigma_r = 1$)	62.8%	62.8%	62.8%	26.483	100

6.1.5 Ionosphere Data Set

Ionosphere data consists of 351 objects with no missing value. There are 33 features and two classes. According to the experiment results in Table 6.7 KPD and KPD-M (G) algorithms perform better than KPD-M (P) and KPD-F algorithms in terms of average accuracy. KPD-M (G) provides 74.5% average accuracy level, which is the highest considering our proposed algorithms, and so close to the accuracy of KFCM-K (G). In addition, this algorithm has small variance compared to KPD algorithms. KPD (G) algorithm finds 74.4% accuracy on the average but with a higher variance. KPD (P) also has a good average accuracy result, which is 74.0%. These two algorithms are better than KFCM-F (G) and KFCM-K (P) algorithms in the literature in this data set.

The best accuracy level is obtained by KPD (P), and the algorithm finds this level 3 times. On the other hand, KPD (G) has the best accuracy of 74.9%, which is achieved 84 times. KPD-M (P) and KPD-M (G) gives 76.9% and 75.5% accuracy, respectively.

Table 6.7: Results for Ionosphere data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 7.07$)	70.7%	70.7%	70.7%	-	-
KFCM-K (G) ($\sigma = 6.32$)	74.6%	74.6%	74.6%	-	-
KFCM-K (P) ($a = 30, b = 8$)	72.7%	72.7%	72.7%	-	-
KPD (P) ($a = 1, b = 12$)	**77.8**%	71.8%	74.0%	0.479	3
KPD (G) ($\sigma = 2$)	**74.9**%	52.7%	**74.4**%	1.334	84
KPD-F (P) ($a = 1, b = 12$)	**77.5**%	62.7%	64.7%	0.200	3
KPD-F (G) ($\sigma = 9$)	65.8%	65.8%	65.8%	0.033	100
KPD-M (P) ($a = 1, b = 12$, $\sigma_r = 2$)	**76.9**%	62.7%	64.7%	11.987	1
KPD-M (G) ($\sigma = 9$, $\sigma_r = 5$)	75.5%	73.2%	**74.5**%	1.639	7

6.1.6 Haberman Data Set

Haberman's Survival data has 306 objects with three features with no missing value. There are two classes which represent whether the patients survived. The experiment results of this data set is given in Table 6.8.

Except KPD-M (G), all the proposed algorithms yield good accuracy results with low variance when compared with the algorithms in the literature. KPD (P) has the highest average accuracy, which is 74.6%. Moreover, it has a low CPU time. The average accuracy of KPD (G) and KPD-F algorithms are 73.5%, and KPD-M (P) has 73.7% accuracy on the average. KPD-F (P) algorithm has no variation. Based on the best accuracy levels, KPD (P) leads the way since it has 75.2%. Then comes KPD-F (G) and KPD-M (P) with 74.2%. Although the KPD-M (P) provides good accuracy, its average CPU time is the highest, which is 44.378 seconds. On the other hand, KPD-M (G) results are not promising.

Table 6.8: Results for Haberman data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 0.71$)	73.8%	73.8%	73.8%	-	-
KFCM-K (G) ($\sigma = 10$)	73.5%	73.5%	73.5%	-	-
KFCM-K (P) ($a = 4, b = 4$)	73.9%	73.9%	73.9%	-	-
KPD (P) ($a = 3, b = 8$)	**75.2**%	73.2%	**74.6**%	0.433	4
KPD (G) ($\sigma = 9.8$)	73.9%	73.4%	73.5%	1.025	1
KPD-F (P) ($a = 0, b = 6$)	73.5%	73.5%	73.5%	0.011	100
KPD-F (G) ($\sigma = 0.01$)	**74.2**%	72.9%	73.5%	0.001	18
KPD-M (P) ($a = 1, b = 12$, $\sigma_r = 10$)	**74.2**%	73.2%	73.7%	44.378	6
KPD-M (G) ($\sigma = 0.02$, $\sigma_r = 20$)	72.5%	57.5%	65.3%	1.503	5

6.1.7 Sonar Data Set

This data set has 208 data points. Each object has 60 features and belongs to one of two clusters. According to the experiment reuslts reported in Table 6.9, KPD-F algorithms performs better on the average accuracy and outperforms KFCM-F and KFCM-K algorithms.

Based on the best accuracy levels obtained in these runs, KPD-M (P), KPD (P), and KPD (G) give the highest best accuracy of 73.1%, 72.6%, and 70.2%, respectively. However these algorithms have high variance, and their average accuracy results are low compared to KPD-F algorithms. Moreover, the best value is observed 3 times only. KPD-F algorithms, on the other hand, observes the best accuracy levels approximately 90 times, and they provide lower variance. All the algorithms have low CPU time since they can perform in less than one second on the average.

Table 6.9: Results for Sonar data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 4$)	-	-	61.4%	-	-
KFCM-K (G) ($\sigma = 10$)	56.3%	56.3%	56.3%	-	-
KFCM-K (P) ($a = 50, b = 2$)	-	-	59.2%	-	-
KPD (P) ($a = 20, b = 2$)	**72.6**%	52.5%	57.2%	0.368	3
KPD (G) ($\sigma = 2$)	**70.2**%	53.8%	56.4%	0.361	3
KPD-F (P) ($a = 40, b = 7$)	63.9%	63.5%	**63.9**%	0.027	94
KPD-F (G) ($\sigma = 3.3$)	62.5%	61.1%	**62.0**%	0.020	96
KPD-M (P) ($a = 50, b = 2$, $\sigma_r = 5$)	**73.1**%	54.9%	59.1%	0.572	3
KPD-M (G) ($\sigma = 8.6$, $\sigma_r = 10$)	63.5%	51.9%	56.3%	0.459	3

6.1.8 SPECT Data Set

SPECT Heart data set has 80 training and 187 test objects. Both training and test data is used in this experiment. There are 22 features and two classes. No missing value is observed. Table 6.10 provides the results of the algorithms on SPECT data.

All of the algorithms perform well when the average accuracy level is considered. Average accuracy level of the proposed algorithms range from 84.1% to 84.6% whereas that of KFCM-F and KFCM-K algorithms vary between 79.4 % and 84.3%. KPD (G), KPD-F algorithms, and KPD-M (G) provide good results with no variation, and KPD-M (P) and KPD-F (G) outperform all other algorithms. KPD-M (P) gives the highest best accuracy 50 times out of 100 trials. KPD (P) has a variance but on the average and best accuracy, it performs well. The average CPU time of all algorithms are low, and that of KPD-F algorithms are the lowest.

Table 6.10: Results for SPECT data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 0.32$)	-	-	80.4%	-	-
KFCM-K (G) ($\sigma = 0.22$)	84.3%	84.3%	84.3%	-	-
KFCM-K (P) ($a = 50, b = 2$)	79.4%	79.4%	79.4%	-	-
KPD (P) ($a = 0, b = 3$)	84.6%	79.4%	84.1%	0.922	35
KPD (G) ($\sigma = 0.02$)	**84.3%**	84.3%	**84.3%**	0.779	100
KPD-F (P) ($a = 5, b = 3$)	84.1%	84.1%	84.1%	0.062	100
KPD-F (G) ($\sigma = 2$)	**84.6%**	**84.6%**	**84.6%**	0.021	100
KPD-M (P) ($a = 30, b = 3, \sigma_r = 10$)	**85.8%**	80.5%	**84.5%**	1.066	50
KPD-M (G) ($\sigma = 0.02, \sigma_r = 0.1$)	**84.3%**	84.3%	**84.3%**	1.023	100

6.2 Synthetic Data Sets

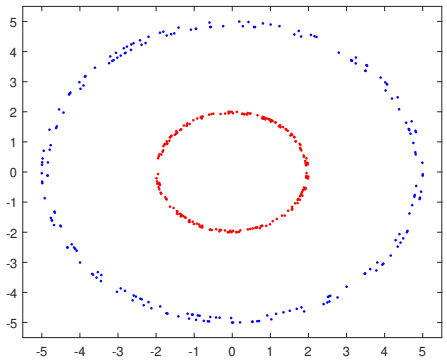
Synthetic data sets are created to measure the performance of the proposed algorithms where the data set cannot be separated linearly. It is difficult to cluster these data sets using traditional clustering algorithm. The experimentation on these data sets reflects the performance of kernel-based clustering on extreme clusters.

Figure 6.1 provides six synthetic data sets that are constructed for this section. These are ring, line, dense, fuzzy X, parabolic, and noisy ring data. Each data set has two clusters, colored by red and blue dots. For the noisy ring, black objects refer to the noise that does not belong to any cluster.

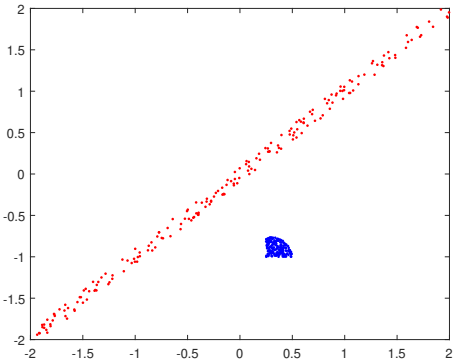
6.2.1 Ring Data Set

In this data set, there is an inner and an outer ring that should be clustered. Each ring consists of 200 data points. Table 6.11 provides the experiment results on ring data set. Note that for each algorithm, number of clusters is taken as two.

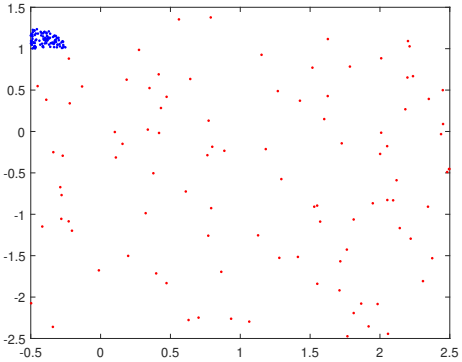
KPD (P) and KPD-M (P) are the two algorithms that can separate the rings and cor-



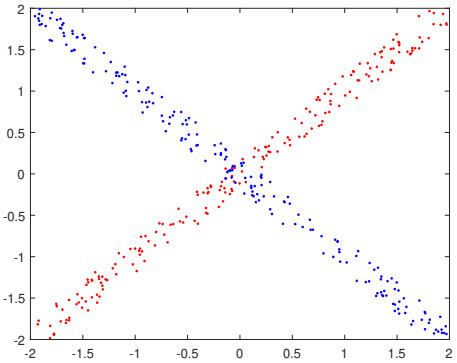
(a) Ring data set



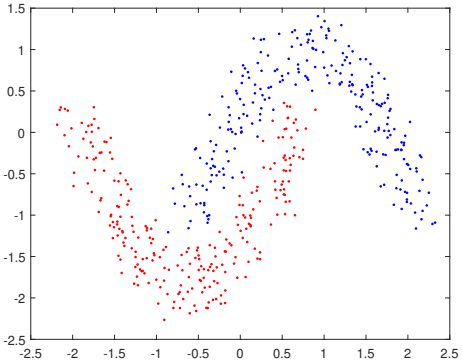
(b) Line data set



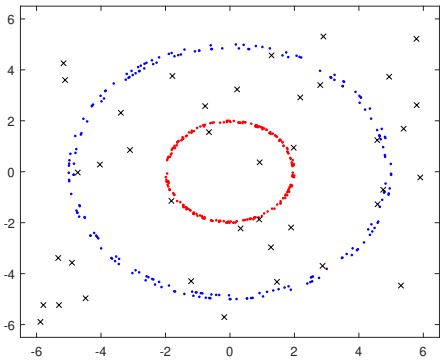
(c) Dense data set



(d) Fuzzy X data set



(e) Parabolic data set



(f) Noisy ring data set

Figure 6.1: Synthetic data sets

rectly clusters all data points with no variation. Based on the CPU time, KPD-M (P) is less costly. The performance of KPD-F algorithms are not promising since they cannot correctly cluster the data set in any of the trials. The KPD (G), on the other hand, can separate the data correctly only 5 times. KFCM-K algorithms in the literature can also provide the best results always while KFCM-F (G) is not successful at separating the rings.

Table 6.11: Results for Ring data set

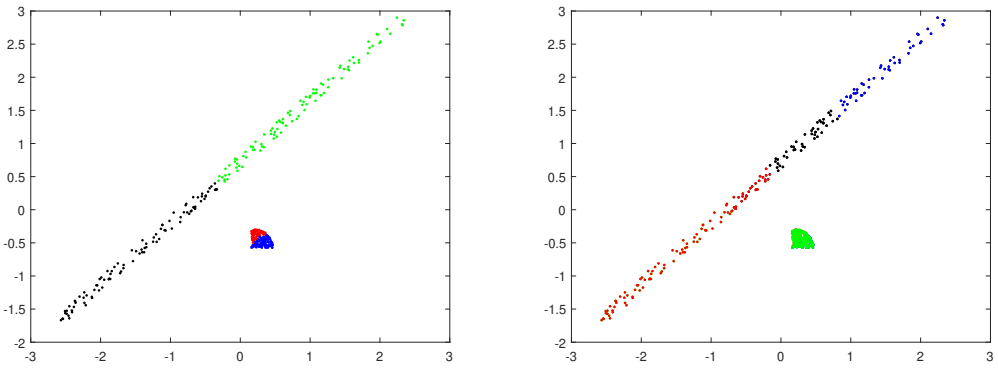
Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 0.22$)	-	-	76.3%	-	-
KFCM-K (G) ($\sigma = 2.83$)	100.0%	100.0%	100.0%	-	-
KFCM-K (P) ($a = 15, b = 8$)	100.0%	100.0%	100.0%	-	-
KPD (P) ($a = 0, b = 4$)	100.0%	100.0%	100.0%	19.233	100
KPD (G) ($\sigma = 0.4$)	100.0%	50.0%	72.7%	0.073	5
KPD-F (P) ($a = 0, b = 2$)	78.3%	50.0%	62.5%	0.044	1
KPD-F (G) ($\sigma = 0.3$)	92.8%	50.0%	63.4%	0.001	4
KPD-M (P) ($a = 1, b = 11$, $\sigma_r = 10$)	100.0%	100.0%	100.0%	8.804	100
KPD-M (G) ($\sigma = 0.4$ $\sigma_r = 10$)	97.0%	52.3%	75.0%	8.230	2

6.2.2 Line Data Set

Line data set has two clusters. The first one is a line, and the second cluster is a sphere-like shape near the line. Each cluster contains 200 objects. Based on the experimental results provided in Table 6.12, kernel fuzzy algorithms obtain 100% accuracy. However, in KFCM-F (G) and KFCM-K (P) algorithms, these results are obtained when the cluster number is set to four.

Therefore, in our experimentation, we first run the algorithms for two-cluster case, which is more accurate considering the behavior of the data set. Then, based on the accuracy results for two clusters, we rerun some of the algorithms by setting the cluster number as three and four in order to improve the clustering results.

All of the proposed algorithms can cluster the line data set with 100% accuracy. KPD



(a) KPD-F (P) result

(b) KPD-F (G) result

Figure 6.2: Results of KPD-F algorithms on line data set when $T = 4$

and KPD-M algorithms can obtain perfect clusters when the cluster number is taken as two. However, the reported results of KPD-F algorithms are when there are four clusters ($T = 4$). Figure 6.2 shows the clusters obtained by KPD-F (P) and KPD-F (G) when T is four. KPD-F (P) divides the line and the sphere into two. KPD-F (G) groups the sphere as one cluster and the line as three clusters. When CPU times of the algorithms are compared, KPD-F algorithms are the fastest while KPD and KPD-M algorithms run slower.

Table 6.12: Results for Line data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 1, T = 4$)	100.0%	100.0%	100.0%	-	-
KFCM-K (G) ($\sigma = 0.71, T = 2$)	100.0%	100.0%	100.0%	-	-
KFCM-K (P) ($a = 30, b = 2, T = 4$)	100.0%	100.0%	100.0%	-	-
KPD (P) ($a = 3, b = 4, T = 2$)	100.0%	100.0%	100.0%	12.454	100
KPD (G) ($\sigma = 0.5, T = 2$)	100.0%	100.0%	100.0%	14.322	100
KPD-F (P) ($a = 10, b = 9, T = 4$)	100.0%	100.0%	100.0%	0.224	100
KPD-F (G) ($\sigma = 1.4, T = 4$)	100.0%	100.0%	100.0%	0.055	100
KPD-M (P) ($a = 4, b = 5, T = 2, \sigma_r = 10$)	100.0%	100.0%	100.0%	12.282	100
KPD-M (G) ($\sigma = 0.9, T = 2, , \sigma_r = 5$)	100.0%	100.0%	100.0%	8.521	100

6.2.3 Dense Data Set

Dense data set has a cluster that is cumulative on the upper left corner and another cluster that spreads uniformly in a range. Each group has 100 objects. According to the results presented in Table 6.13, KFCM-K (G) algorithm can cluster the dense data with 100% accuracy using two clusters only. KFCM-K (P) and KFCM-F (G) obtains 99.5% and 98.8% accuracy, respectively, and KFCM-F (G) has a variation. In addition, these two algorithms obtain these accuracy levels when the cluster number is set to six.

From the proposed algorithms, KPD and KPD-M algorithms with both polynomial and Gaussian kernels yield 100% accuracy when they are run for two clusters. The CPU time of these algorithms are close to each other and less than one second except KPD (P) algorithm. For KPD-F (P), the best accuracy level is observed when number of clusters is fixed to three. With this way, KPD-F (P) correctly clusters the dense data. On the other hand, KPD-F (G) algorithm gives its best result when the cluster number is set to four. On the average, it provides 96.6% accuracy, and it reaches 100% accuracy only once. KPD-F algorithms are the fastest ones when their CPU time is considered.

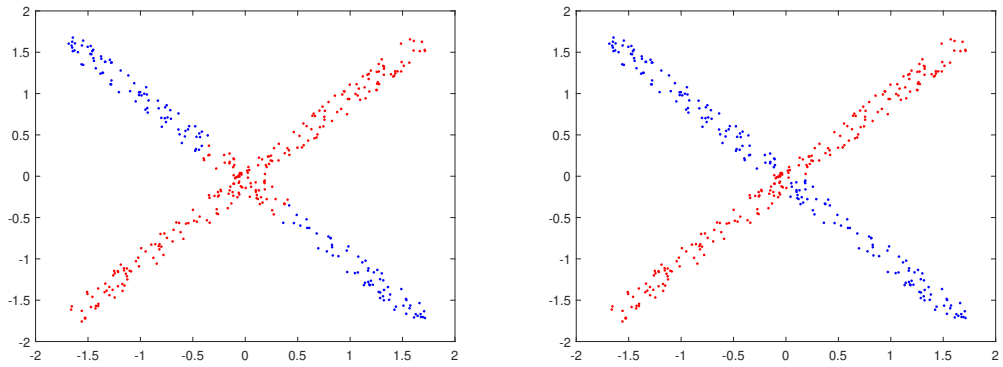
Table 6.13: Results for Dense data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 7.07, T = 6$)	-	-	98.8%	-	-
KFCM-K (G) ($\sigma = 0.32, T = 2$)	100.0%	100.0%	100.0%	-	-
KFCM-K (P) ($a = 2, b = 4, T = 6$)	99.5%	99.5%	99.5%	-	-
KPD (P) ($a = 0, b = 3, T = 2$)	100.0%	100.0%	100.0%	1.180	100
KPD (G) ($\sigma = 0.2, T = 2$)	100.0%	100.0%	100.0%	0.620	100
KPD-F (P) ($a = 2, b = 7, T = 3$)	100.0%	100.0%	100.0%	0.070	100
KPD-F (G) ($\sigma = 0.8, T = 4$)	100.0%	91.0%	96.6%	0.013	1
KPD-M (P) ($a = 3, b = 5, T = 2, \sigma_r = 10$)	100.0%	100.0%	100.0%	0.571	100
KPD-M (G) ($\sigma = 0.3, T = 2, \sigma_r = 1$)	100.0%	100.0%	100.0%	0.586	100

6.2.4 Fuzzy X Data Set

Fuzzy X data is one of the most challenging data sets since objects from two clusters intersect with each other in the middle. Therefore, it would be difficult to separate the clusters. The experiments are reported for two clusters only since increasing the number of clusters does not affect the accuracy levels. Table 6.14 provides the accuracy levels of each algorithm.

Based on the average accuracy levels, KPD-M (P) outperforms all of the algorithms. However, it has a higher variance. The average accuracy value of KPD-F (P) provides the second highest accuracy, and KPD-F (P) has a lower variance when compared with all other algorithms. From the kernel-based fuzzy algorithms, KFCM-K (P) gives the highest accuracy on the average, which is 75.6%. KFCM-K (G) is the one with the lowest average accuracy level when compared with all the algorithms. When the best acuracy is examined, KPD-M (P) provides the highest result 11 times, which is 96.8%. Then comes KPD-M (G) and KPD algorithms. Figure 6.3 shows the clusters obtained by KPDM (P) algorithms with best and average accuracy levels, respectively.



(a) Average KPD-M (P) result

(b) Best KPD-M (P) result

Figure 6.3: Results of KPD-F algorithms on line data set when $T = 4$

We can conclude that considering the average results, Mahalanobis distance has a distinguished effect on explaining the clusters. However, KPD-F (P) is the most consistent algorithm since it performs the best result almost half of the trials.

Table 6.14: Results for Fuzzy X data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 1.41$)	-	-	69.5%	-	-
KFCM-K (G) ($\sigma = 1.41$)	-	-	52.0%	-	-
KFCM-K (P) ($a = 0, b = 2$)	-	-	75.6%	-	-
KPD (P) ($a = 8, b = 11$)	87.8%	50.3%	61.1%	3.954	4
KPD (G) ($\sigma = 0.2$)	86.8%	50.0%	58.4%	7.630	2
KPD-F (P) ($a = 0, b = 2$)	76.8%	75.0%	**75.8**%	0.027	48
KPD-F (G) ($\sigma = 0.8$)	73.5%	50.5%	66.5%	0.023	8
KPD-M (P) ($a = 0, b = 2$, $\sigma_r = 2$)	**96.8**%	56.0%	**86.5**%	8.515	11
KPD-M (G) ($\sigma = 0.3$, $\sigma_r = 5$)	89.0%	50.0%	55.2%	8.511	2

6.2.5 Parabolic Data Set

Parabolic data set consists of two parabolas having some randomness. Each parabola contain 250 data points. The experiments are conducted for two clusters, and Table 6.15 reports the results.

From all of the algorithms KFCM-K (G) has the highest average accuracy, which is 89.0%. The average accuracy level of the proposed algorithms ranges between 84.3-87.7%. KPD-F (P) provides an average accuracy level of 87.7% with a less deviation. Best accuracy level is obtained by KPD (G) algorithm 31 times. KPD-M (G) gives the second highest best accuracy, but performing it three times only. KPD-F algorithms have less variation when compared with KPD and KPD-M algorithms. CPU times of KPD algorithms are the highest and those of KPD-F give the lowest.

Briefly, the average results of the algorithms on the parabolic data are similar to each other while KPD-F algorithms have a low variance and kernel-based fuzzy algorithms has no variance.

Table 6.15: Results for Parabolic data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KFCM-F (G) ($\sigma = 1$)	88.2%	88.2%	88.2%	-	-
KFCM-K (G) ($\sigma = 1$)	89.0%	89.0%	89.0%	-	-
KFCM-K (P) ($a = 10, b = 12$)	87.8%	87.8%	87.8%	-	-
KPD (P) ($a = 2, b = 3$)	88.4%	81.4%	86.0%	26.665	5
KPD (G) ($\sigma = 0.6$)	**90.4**%	71.2%	86.4%	19.238	31
KPD-F (P) ($a = 7, b = 7$)	87.8%	87.0%	87.7%	0.182	63
KPD-F (G) ($\sigma = 1.2$)	87.2%	86.8%	86.9%	0.053	60
KPD-M (P) ($a = 1, b = 3$, $\sigma_r = 10$)	88.6%	72.0%	84.3%	13.257	2
KPD-M (G) ($\sigma = 0.9$, $\sigma_r = 10$)	**89.2**%	77.8%	85.8%	13.668	3

6.2.6 Noisy Ring Data Set

This data set is created by adding uniform noise to ring data set. Number of noises is determined as the 10% of the number of objects in the ring data. It is assumed that noises does not have any cluster. Cluster accuracy is measured whether the objects in the rings are correctly clustered. Since KPD (P) and KPD-M (P) provide the best accuracy in the ring data set, noisy ring data set is run for these two algorithms. Table 6.16 refers to the experiment results. Both of the algorithm correctly clusters the rings and do not affect on the noises at all. In terms of CPU time, KPD-M (P) is faster than KPD (P) algorithm.

Table 6.16: Results for Noisy Ring data set

Algorithms	Best	Worst	Average	Avg CPU Time (sec)	Best Hit
KPD (P) ($a = 0, b = 4$)	100.0%	100.0%	100.0%	23.660	100
KPD-M (P) ($a = 1, b = 11$, $\sigma_r = 10$)	100.0%	100.0%	100.0%	10.105	100

6.3 Comments on the Performance of the Proposed Algorithms

6.3.1 Performance on the Real Data Sets

When the performances of proposed algorithms for real data sets are compared, it can be seen that KPD-F (G) has the lowest variance in general. It also provides good results in terms of average accuracy in many real data sets. The performance of KPD-F (P) depends on the data sets. Although in some data sets KPD-F (G) performs well when it comes to best accuracy level, its performance is not steady.

KPD algorithms yield good results in terms of best accuracy. Their performances for the average accuracy depend on data sets. They have high variance when compared with KPD-F algorithms. Performance of KPD algorithms are promising when their best accuracy levels are examined; therefore, they require good initialization to reduce the variance of the results and increase the average accuracy levels.

As in KPD algorithms, KPD-M algorithms suffer from the initialization, so they do not perform well in terms of average accuracy levels. However, according to their best accuracy levels, they produce good results. Thus, improving the initialization will affect the performance of KPDM-algorithms positively.

When the average CPU time of the algorithms are compared, KPD-F algorithms are the fastest. In all of the data sets, they give results in less than a second. The speed of KPD and KPD-M algorithms depends on the size of the data sets. When the number of data points and features are increased, their average CPU time is adversely affected.

6.3.2 Performance on the Synthetic Data Sets

KPD algorithms perform well on synthetic data sets. In most of the cases, KPD (P) outperforms KPD (G). These algorithms provide good results when the number of clusters is selected as two. In addition, KPD (P) is experimented for noise data, and it is observed that KPD (P) is noise-insensitive.

KPD-M algorithms work well on almost all synthetic data sets. KPD-M (P) algorithm outperforms the other algorithms in data sets with intersecting clusters. In addition,

KPD-M algorithms yield good results for two clusters. The performance of KPD-M (P) algorithm on the noisy data shows that it is not sensitive to the noises.

When the performance of KPD-F algorithms are compared, KPD-F (P) is better than KPD-F (G) in terms of average accuracy. In some data sets, KPD-F algorithms performs well when the number of clusters are increased. KPD-F algorithms have the lowest average CPU times than KPD and KPD-M algorithms in synthetic data.

CHAPTER 7

CONCLUSION

Clustering is an unsupervised machine learning technique that aims to group the similar objects and to separate dissimilar objects from each other. While doing so, it considers an objective function and an appropriate similarity measure.

Kernel-based clustering methods are used to group the non-linearly separable data sets in a higher dimensional space. In this space, it is assumed that the data can be linearly separable, and traditional clustering approaches can be applied there. However, it is computationally costly to map the data into a high-dimensional space and make a calculation in that space. Kernel functions solve this problem by allowing us to obtain calculation results in higher dimensional space using the non-mapped (original) data set. That is, there is no need to obtain the mapped data points to cluster them in the higher dimensional space. Clustering in that space can also be made using the objects in the original space with the help of kernel functions.

In this study, we work on a kernel-based soft clustering approach. Our focus is Probabilistic Distance Clustering (PDC) given in [2], which is a soft clustering approach that has a basic principle saying that membership probability of an object and its distance to a cluster are inversely proportional. We adapt kernel method to PDC approach, and propose three novel kernel-based PDC algorithms to the literature.

The principles of kernel-based PDC has been developed based on PDC principles. Then using those principles, optimization models for this problem are generated. We propose three novel algorithms. The first algorithm, KPD, defines and updates the cluster centers in the kernel space implicitly. Center representation in KPD has a novel approach. Centers are formed as the convex combination of all objects, and

they are updated by updating the center coefficients. Moreover, an algorithm to obtain the approximate centers in the input space is provided. The second algorithm, KPD-F, constructs the cluster centers in the input space and applies clustering by mapping the centers into feature space. Both KPD and KPD-F algorithms use Euclidean distance.

We also propose two kernelized Mahalanobis distance functions for feature space. These functions differ depending on whether the covariance matrix is invertible or not. Please note that the kernelized Mahalanobis distance functions in the literature are developed either for squared Mahalanobis distance or by regularizing the objective function of the clustering problem. The proposed Mahalanobis distance functions, on the other hand, do not use these approaches in the literature. Given the covariance matrix is non-invertible, we develop KPD-M algorithm, which is a kernel-based Mahalanobis PDC method. This algorithm follows the center definition and update rule as in KPD algorithm considering the proposed Mahalanobis distance.

Experimental study for measuring the performance of these algorithms has been conducted. We test the algorithms on both real and synthetic data sets, and compare the results with soft kernel-based clustering algorithms. The performance of the algorithms in the real data sets are promising. When the best accuracy levels are examined, proposed algorithms generally outperform soft clustering methods. On the other hand, average accuracy results show that algorithms depend on the initialization for some data sets. For synthetic data, our algorithms work with 100% accuracy in many data sets with requiring less number of clusters.

Kernel-based PDC algorithms suffer from initialization since the results are strongly dependent on the initialization, especially for real data set. As a future research direction, the initialization effect should be reduced. In addition, proposed algorithms are developed assuming that cluster sizes are given as fixed. However, data sets may have different cluster sizes, and taking the cluster size as fixed may affect the performance of the clustering algorithm adversely. Therefore, as another future research, enhancing these algorithms for different cluster sizes should be considered.

The proposed algorithms are designed for a single kernel. Instead of using a single kernel function, multiple kernel functions can be used by defining the kernel function as a linear combination of multiple kernel functions. As a future work, kernel-based

PDC with multiple kernels can be studied.

REFERENCES

[1] M. Goebel and L. Gruenwald, "A survey of data mining and knowledge discovery software tools," *ACM SIGKDD explorations newsletter*, vol. 1, no. 1, pp. 20–33, 1999.

[2] A. Ben-Israel and C. Iyigun, "Probabilistic d-clustering," *Journal of Classification*, vol. 25, no. 1, pp. 5–26, 2008.

[3] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM computing surveys (CSUR)*, vol. 31, no. 3, pp. 264–323, 1999.

[4] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, 2017.

[5] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 281–297, Oakland, CA, USA, 1967.

[6] L. Rousseeuw and P. Kaufman, "Clustering by means of medoids," in *Proceedings of the statistical data analysis based on the L1 norm conference, neuchatel, switzerland*, vol. 31, 1987.

[7] C. Iyigun and A. Ben-Israel, "Probabilistic distance clustering adjusted for cluster size," *Probability in the Engineering and Informational Sciences*, vol. 22, no. 4, pp. 603–621, 2008.

[8] C. Iyigun and A. Ben-Israel, "Semi-supervised probabilistic distance clustering and the uncertainty of classification," in *Advances in data analysis, data handling and business intelligence*, pp. 3–20, Springer, 2009.

[9] Y. Caner, "Two-mode probabilistic distance clustering," Master's thesis, Middle East Technical University, 2021.

[10] J. C. Bezdek, *Fuzzy-Mathematics in Pattern Classification.* Cornell University, 1973.

[11] J. C. Bezdek, R. Ehrlich, and W. Full, "Fcm: The fuzzy c-means clustering algorithm," *Computers & geosciences*, vol. 10, no. 2-3, pp. 191–203, 1984.

[12] D. E. Gustafson and W. C. Kessel, "Fuzzy clustering with a fuzzy covariance matrix," in *1978 IEEE conference on decision and control including the 17th symposium on adaptive processes*, pp. 761–766, IEEE, 1979.

[13] I. Gath and A. B. Geva, "Unsupervised optimal fuzzy clustering," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 773–780, 1989.

[14] M. Reed, *Methods of modern mathematical physics: Functional analysis*. Elsevier, 2012.

[15] B. Schölkopf, A. J. Smola, F. Bach, *et al.*, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[16] S. Y. Kung, *Kernel methods and machine learning*. Cambridge University Press, 2014.

[17] M. A. Aizerman, "Theoretical foundations of the potential function method in pattern recognition learning," *Automation and remote control*, vol. 25, pp. 821–837, 1964.

[18] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[19] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem," *Neural computation*, vol. 10, no. 5, pp. 1299–1319, 1998.

[20] M. Girolami, "Mercer kernel-based clustering in feature space," *IEEE transactions on neural networks*, vol. 13, no. 3, pp. 780–784, 2002.

[21] D.-Q. Zhang and S.-C. Chen, "Kernel-based fuzzy and possibilistic c-means clustering," in *Proceedings of the international conference artificial neural network*, vol. 122, pp. 122–125, 2003.

[22] Z.-d. Wu, W.-x. Xie, and J.-p. Yu, "Fuzzy c-means clustering algorithm based on kernel method," in *Proceedings Fifth International Conference on Computational Intelligence and Multimedia Applications. ICCIMA 2003*, pp. 49–54, IEEE, 2003.

[23] D.-Q. Zhang and S.-C. Chen, "A novel kernelized fuzzy c-means algorithm with application in medical image segmentation," *Artificial intelligence in medicine*, vol. 32, no. 1, pp. 37–50, 2004.

[24] H. Shen, J. Yang, S. Wang, and X. Liu, "Attribute weighted mercer kernel based fuzzy clustering algorithm for general non-spherical datasets," *Soft Computing*, vol. 10, no. 11, pp. 1061–1073, 2006.

[25] Y. Ding and X. Fu, "Kernel-based fuzzy c-means clustering algorithm based on genetic algorithm," *Neurocomputing*, vol. 188, pp. 233–238, 2016.

[26] Z. Li, S. Tang, J. Xue, and J. Jiang, "Modified fcm clustering based on kernel mapping," in *Object Detection, Classification, and Tracking Technologies*, vol. 4554, pp. 241–245, SPIE, 2001.

[27] S. Zhou and J. Q. Gan, "Mercer kernel, fuzzy c-means algorithm, and prototypes of clusters," in *International Conference on Intelligent Data Engineering and Automated Learning*, pp. 613–618, Springer, 2004.

[28] D. Z. S. Chen, "Fuzzy clustering using kernel method," *IEEE, Nanjing, China*, 2002.

[29] D. Graves and W. Pedrycz, "Kernel-based fuzzy clustering and fuzzy clustering: A comparative experimental study," *Fuzzy sets and systems*, vol. 161, no. 4, pp. 522–543, 2010.

[30] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta, "A survey of kernel and spectral methods for clustering," *Pattern recognition*, vol. 41, no. 1, pp. 176–190, 2008.

[31] N. Baili and H. Frigui, "Fuzzy clustering with multiple kernels," in *2011 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2011)*, pp. 490–496, IEEE, 2011.

[32] I. Dagher, "Fuzzy clustering using multiple gaussian kernels with optimized-parameters," *Fuzzy Optimization and Decision Making*, vol. 17, no. 2, pp. 159–176, 2018.

[33] H.-C. Huang, Y.-Y. Chuang, and C.-S. Chen, "Multiple kernel fuzzy clustering," *IEEE Transactions on Fuzzy Systems*, vol. 20, no. 1, pp. 120–134, 2011.

[34] S. Zeng, X. Wang, X. Duan, S. Zeng, Z. Xiao, and D. Feng, "Kernelized mahalanobis distance for fuzzy clustering," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 10, pp. 3103–3117, 2020.

[35] B. Schölkopf, S. Mika, C. J. Burges, P. Knirsch, K.-R. Muller, G. Ratsch, and A. J. Smola, "Input space versus feature space in kernel-based methods," *IEEE transactions on neural networks*, vol. 10, no. 5, pp. 1000–1017, 1999.

[36] B. Haasdonk and E. Pękalska, "Classification with kernel mahalanobis distance classifiers," in *Advances in Data Analysis, Data Handling and Business Intelligence*, pp. 351–361, Springer, 2009.

[37] D. Dua and C. Graff, "UCI machine learning repository." `http://archive.ics.uci.edu/ml`, 2017.

[38] O. L. Mangasarian and W. H. Wolberg, "Cancer diagnosis via linear programming," tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1990.