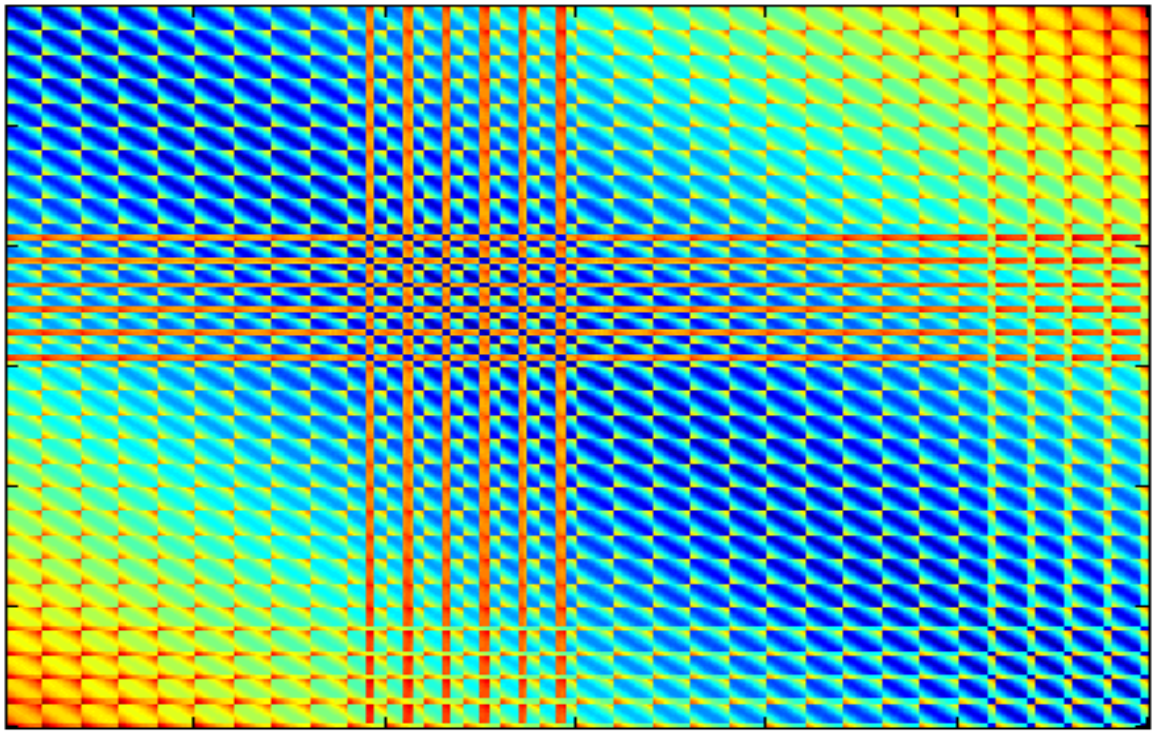
UNIVERSITY OF TROMSØ UIT

FACULTY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF PHYSICS AND TECHNOLOGY

Mean Shift Spectral Clustering Using Kernel Entropy Component Analysis



Jørgen Andreas Agersborg

FYS-3921 Master's Thesis in Electrical Engineering

June 2012

Abstract

Clustering is an unsupervised pattern recognition technique for finding natural groups in data, whether it is a grouping of web pages found by a search engine or segmenting satellite images into different types of ground cover. There exists a variety of different ways to perform clustering ranging from heuristics rules designed for a specific dataset to general procedures which can be applied to all datasets with varying degrees of success. The k-means algorithm is a well known example of the latter approach that can be expected to give good results when the data is easily separable.

Another example of general clustering procedures are spectral clustering methods which involve a non-linear data transformation that allows them to handle complex cluster structures. They are considered to be the state of the art in the clustering literature, but are computationally demanding and unable to handle large datasets. To overcome the size limitations, this thesis uses a two-stage approach. The first stage can be seen as a preprocessing to reduce the size of the input for the spectral clustering in the second stage.

The preprocessing is accomplished by using a simple clustering method on the dataset. These clusters represents a partitioning and is a more natural way of representing the dataset than randomly selecting a subset of samples. One can then adjust the number of partitions so spectral clustering methods can be used.

The procedure is called Mean Shift Spectral Clustering (MSSC) as the mean shift clustering algorithm is used in the first stage. Each partition found by mean shift consists of data points that are close to the same mode in the estimated probability density function. Hence the partitions will be representative of the geometric structure of the dataset.

This thesis realizes for the first time the idea of spectral clustering based on the partitions found by mean shift. The Kernel Entropy Component Analysis (KECA) spectral clustering method, a recent development in the field of spectral clustering, is used for this purpose and compared with the better known Kernel Principal Component Analysis (KPCA) method.

A comprehensive collection of MATLAB functions has been developed to allow the testing of this procedure which is able to handle large and high dimensional datasets. It is able to cluster images directly with as many feature vectors as there are pixels. The experiments show how the clustering accuracy varies as a function of the primary parameters. This gives a good overall characterization of the method and what can be expected when used for unfamiliar datasets.

The MSSC procedure is shown to provide good clustering results when following some basic principles for selecting parameters.

Acknowledgments

First of all I would like to start by thanking my supervisor, Robert Jenssen. Your inspiring lectures and enthusiasm directed my studies towards signal processing and further into the field of pattern recognition, a choice I have never regretted. The feedback during the process of writing this thesis has always been very constructive and has helped keep me on track.

My parents have provided much appreciated support by occasionally bringing me food at the office. In hectic periods this has given me much needed relief from frozen pizzas and dinners that only require a microwave or boiling water. The latest visits brought large amounts of fruit, presumably because you feared that my diet might cause me to develop scurvy.

To Vidar, I appreciate your interesting dilemmas, clustering discussions and general tomfoolery. Also, to the rest of the people at Norutbrakka, I am thankful for the pleasure of your company. A big thanks to Thomas and Jonas who have read through the thesis, particularly for the extensive feedback from the latter. Kristine, thank you for giving me motivation and keeping me company in the final stages of the writing.

Finally, to the two cows seen in several plots in this thesis; I don't know who you are, but I'm sure you'll appear in many brightly coloured dreams and nightmares for a long time to come. If I ever have to segment an image with a cow again, it will be too soon.

Jørgen Agersborg
Tromsø, June 2012

Contents

1 Introduction **1**
- 1.1 Structure of this Thesis . . . 6
- 1.2 Notation . . . 7

2 Some Clustering Procedures **11**
- 2.1 Introduction . . . 11
  - 2.1.1 Types of Clustering Procedures . . . 11
  - 2.1.2 Methods and Algorithms Presented . . . 12
- 2.2 The K-Means Algorithm . . . 13
  - 2.2.1 Generalized Hard Algorithm Scheme . . . 13
  - 2.2.2 The K-Means Algorithm as a Special Case of GHAS . 16
- 2.3 EM with Gaussian Mixture Models . . . 24
  - 2.3.1 Mixture Models . . . 24
  - 2.3.2 Estimating Gaussian Mixture Model Parameters . . . 28
  - 2.3.3 Clustering by EM of Mixture Parameters . . . 33
- 2.4 The Mean Shift Algorithm . . . 35
  - 2.4.1 Kernel Density Estimation . . . 36
  - 2.4.2 Mean Shift Algorithm Variations . . . 39
- 2.5 Kernel PCA Spectral Clustering . . . 46
  - 2.5.1 Principal Component Analysis . . . 46
  - 2.5.2 The Kernel Trick . . . 49
  - 2.5.3 Kernel PCA . . . 50
- 2.6 Laplacian Eigenmaps . . . 56
  - 2.6.1 Defining The Graph . . . 57
  - 2.6.2 Graph Laplacian . . . 60
  - 2.6.3 Laplacian Clustering Methods . . . 68

3 Information Theoretical Learning **71**
- 3.1 Introduction . . . 71
- 3.2 Information . . . 72
- 3.3 Shannon Entropy . . . 73

3.4 Renyi Entropy . . . 75
3.5 Entropy in Physics . . . 77
3.6 Divergence . . . 79

4 Kernel Entropy Component Analysis **81**
4.1 Introduction . . . 81
4.2 Estimating Renyi's Quadratic Entropy . . . 81
4.3 The KECA Transformation . . . 83
4.4 KECA Spectral Clustering . . . 85

5 Novel Mean Shift Spectral Clustering **93**
5.1 Introduction . . . 93
5.2 A Two-Stage Clustering Approach . . . 94
5.3 The First Stage . . . 94
5.4 The Second Stage . . . 96
5.5 Proposed Changes in MSSC . . . 97
5.6 Parameters in Two-Stage Clustering . . . 98

6 Results **101**
6.1 Examples to Illustrate MSSC . . . 101
6.1.1 The Iris Dataset . . . 101
6.1.2 The Cows Image . . . 108
6.1.3 Toy Data Example . . . 116
6.2 Image Segmentation . . . 119
6.2.1 Plane Picture . . . 120
6.2.2 Water Buffalo Picture . . . 121
6.3 Additional Datasets . . . 123
6.3.1 Wisconsin Breast Cancer Dataset . . . 123
6.3.2 Wine Dataset . . . 125

7 Conclusion **127**
7.1 Suggestions for Further Work . . . 128
7.1.1 General Suggestions . . . 128
7.1.2 First Stage . . . 129
7.1.3 Second Stage . . . 130

Chapter 1

Introduction

The information age has brought with it enormous amounts of easily accessible digital data where meaningful groups can be found. Organizing similar objects in groups has been an important scientific technique for a long time, where a classic example is categorization of living organisms into taxonomic ranks in biology. Other prominent examples of grouping data includes image segmentation [7], web page organization [38], characterizing genome sequences [44], customer type grouping for market research [3] and social network analysis [45]. Yet another application of clustering is data compression for organizing the data and summarizing it by cluster prototypes [16].

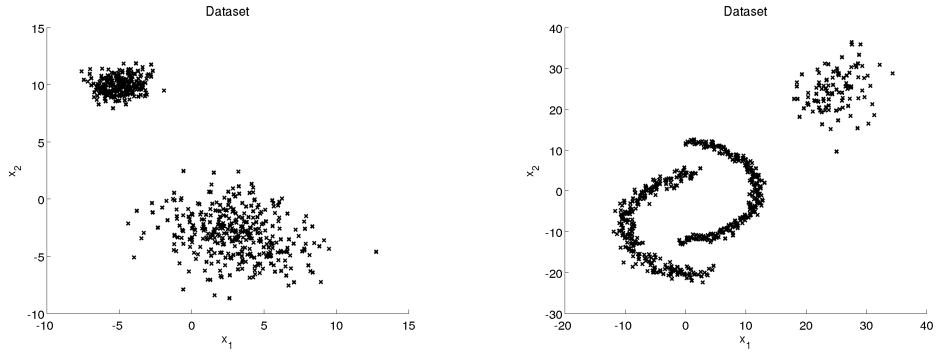
Often data is represented by many variables and is therefore high dimensional. While humans are excellent at finding clusters in two and possibly three dimensions by visual inspection, automatic algorithms are needed for high dimensional data. Even in two and three dimensions, manual group assignments as an overall strategy is not feasible once time and cost are taken into account. A better approach is to use computers to find the group structure and then let humans interpret what they mean.

Clustering is an unsupervised pattern recognition technique which automatically seeks to gather objects in "natural" groups [20, Chapter 12]. In addition to providing a natural classification, clustering gives an insight into the underlying structure of the data. The goal is that the objects in each cluster are *similar* while the different clusters are *dissimilar*. To do this we need to define a *proximity measure* that quantifies what we mean by these terms. A proximity measure is either a similarity measure or a dissimilarity measure, where a much used example of the latter is the *Euclidean distance*.

The different ways of defining proximity is part of the reason why many different clustering procedures have been developed, ranging from simple heuristics suitable for a particular type of dataset to general iterative schemes which seeks to optimize some associated optimality criterion. Ideally one

should use a clustering approach that produce good results in a wide variety of situations, since the general assumption for clustering is that we know little or nothing about the data in advance.

A well known general clustering procedure is the k-means algorithm [22]. This is most often implemented with the Euclidean distance[1]. Given a set of cluster representatives, in the first step each point is assigned to its closest representative. The second step updates the cluster representatives by setting them equal to the mean of the data vectors assigned to them in the previous step. These two steps are repeated until some convergence criterion is met, see Section 2.2 for details.



(a) Dataset 1. (b) Dataset 2.

Figure 1.1: Two datasets.

The k-means algorithm is optimal for clustering dense, spherically shaped and linearly separable clusters [16]. Fig. 1.1a shows an example of such a situation, and the k-means algorithm can be expected to give a good clustering results.

This is not the case if we assume that in addition to the points in the upper right corner, each of the two half circles in (b) should be different clusters. Clearly, while it is easy to separate the corner cluster from the other two, it is not possible to define a straight line that separates the two half circles.

Figure 1.2 shows the clustering result of the k-means algorithm for three clusters. This is not a good result, but is it possible solve this without defining an ad hoc decision curve between the two half circles?

This is where spectral clustering methods excel. By performing a non-linear data transformation, the data is transformed to some space where it

[1]Or the squared Euclidean distance, in which case the k-means algorithm can be derived from optimizing the associated cost function as will be shown in Section 2.2.1.

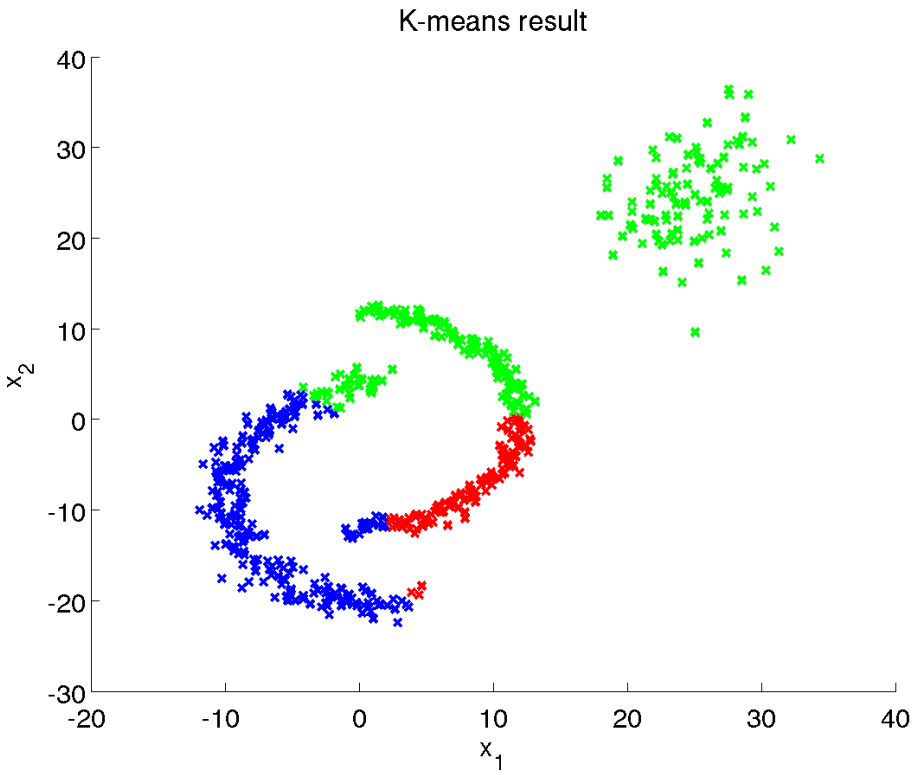


Figure 1.2: K-means solution.

is easier to separate between the clusters. Since they are able to handle non-linearities as well as easier datasets such as in Fig. 1.1a, spectral clustering methods are considered to be the state of the art [28] and they often outperforms traditional clustering algorithms such as k-means [41]. An introduction to spectral methods can be found in [26] while [36] introduces a graph-based spectral algorithm for image segmentation problems.

Spectral methods are based on finding the eigenvalues and eigenvectors of a data affinity matrix which contains *proximity measures* between all pairs of data points. However, this means that spectral methods are not suitable for large datasets, since the size of the matrix increases as the number of data points squared.

When clustering large datasets, such as images, one often has to choose between using a simple clustering procedure on the entire set or using an advanced method on a subset of the original data and then use some heuristic to assign the rest of the data to the clusters found. Needless to say, the final result will then depend both on the subset used and the method for assigning the other points to the clusters.

In this thesis we propose an alternative to this dilemma. The idea is to use a two-stage clustering approach where the first stage can be seen as a form of preprocessing to the reduce the size of the input to the spectral clustering method in the second stage. The method is called Mean Shift Spectral Clustering (MSSC), as the mean shift clustering algorithm [6, 12] is used in the first stage.

Thus the preprocessing in the first stage is performed by using a simple[2] clustering method on the dataset. Since the mean shift algorithm is based on estimating the underlying probability density function, this initial clustering represents a natural partitioning of the dataset. Then in the second stage, spectral clustering is performed on an affinity matrix based on these partitions. Using proximity measures based on information theory (see Chapter 3) the entire dataset is used when constructing this matrix.

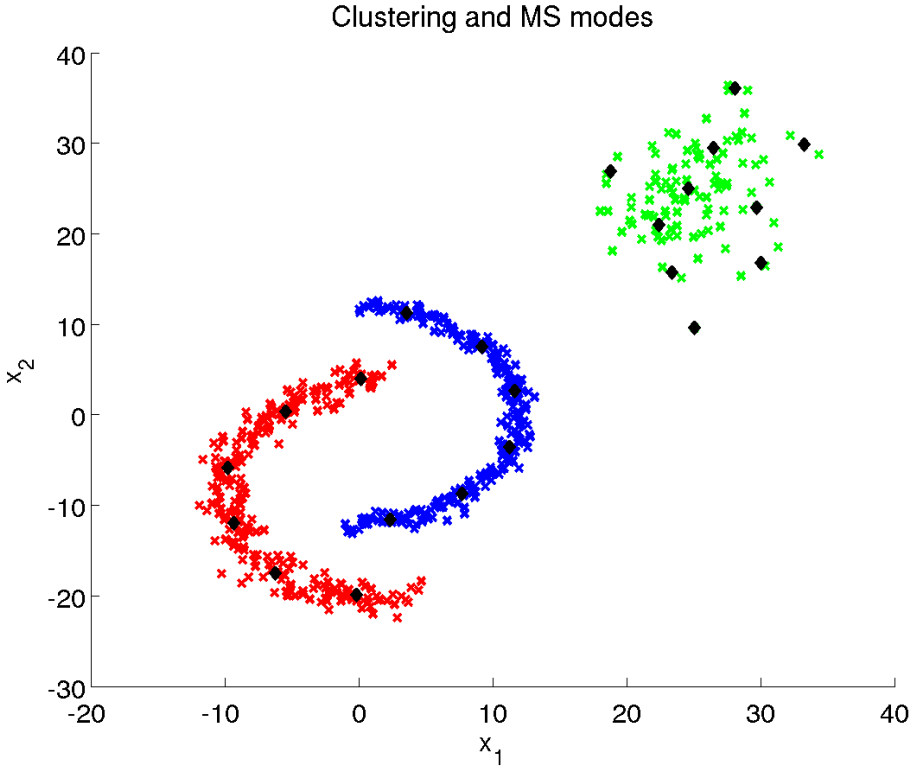


Figure 1.3: MSSC solution.

Figure 1.3 shows the clustering result by using MSSC on the dataset seen

[2]While the mean shift algorithm used in MSSC is not really considered "simple" compared to many other clustering procedures, it can be used on much larger datasets than spectral methods.

in Fig. 1.1b. The black dots show the center of the partitions found by mean shift. While useful for large datasets, MSSC may also improve performance in cases where one could have used spectral methods directly. This we shall see when we return to this example in Section 6.1.3.

This thesis also introduces a very recent development in the field of clustering, the Kernel Entropy Component Analysis (KECA) spectral clustering method, into the MSSC framework. KECA is based on preserving entropy and was first presented in [18] motivated by trying to shift the focus of clustering procedures away from various forms of normality assumptions. Its performance in this setting is compared with the well known spectral clustering method Kernel Principal Component Analysis [14, 25].

An extensive framework of code has been developed and used to illuminate many aspects of MSSC. This has allowed the method to be tested on some well known pattern recognition benchmarks, with the focus being on illustrating how the different parameter choices affects the clustering result. This was considered more important than simply presenting the best result as one generally does not know the true labels in a clustering setting, thus limiting the effectiveness of tweaking parameters. A two-stage approach will significantly increase the number of possible parameter combinations and results will be presented based on combining variations of the most important ones.

The experiments have a particular focus on the effect of the kernel size. This is a parameter which is hard to optimize. There is no commonly accepted approach to selecting the kernel size that do not involve making certain assumptions about the dataset.

The idea of a two-stage approach was first presented in [28]. While the article named it Mean Shift Spectral Clustering, the second stage in the method presented was not actually spectral clustering, see Section 5.4. It was based on the partition affinity matrix, but used a heuristic approach resembling hierarchical clustering with a much higher computational complexity than spectral methods.

Novel improvements developed in this thesis include:

- Performing actual spectral clustering based on the mean shift partitions.
- Introducing the KECA spectral clustering method, a recent development in the field in MSSC and comparing it with the better known KPCA, which is also new in the MSSC setting.
- Having different kernel size parameters in first and second stage to emphasize the different roles of the two stages.

- While having different kernel sizes eliminates the need to save the kernel matrix from the first stage, we introduce the *option* of storing the associated distance matrix. For many kernel functions this allows the construction of the kernel matrix based on calculations done in mean shift with a different bandwidth.

- Using the option to blur (see Section 2.4.2) in the mean shift algorithm to reduce its sensitivity to the kernel size.

1.1 Structure of this Thesis

A selection of clustering procedures are discussed in Chapter 2. Two spectral methods relevant for the second stage and three non-spectral methods which can be used in the first stage are presented. For the benefit of the reader the clustering procedures are discussed thoroughly. Each section contains the motivation behind the approach and how to cluster with the procedure as well as figures to illustrate the performance. Kernels are briefly introduced in connection with the non-parametric probability density estimation approach Parzen windowing, which is used for the mean shift algorithm in Section 2.4. The concept of kernels are further expanded in Section 2.5 when used to find the kernel Principal Component Analysis (KPCA) data transformation.

Information Theoretical Learning (ITL), which introduces the use of information theory descriptors such as entropy and divergence for a wide range of applications, is presented in Chapter 3. The chapter starts with the concept of information, the Shannon entropy and the connection with entropy in physics to provide a brief introduction into this comprehensive field. ITL however focuses on the Renyi entropy in Section 3.4 and the associated Cauchy-Schwarz divergence measure found in Section 3.6.

These ITL descriptors are then used in Chapter 4 for defining KECA, which is based on preserving the estimated quadratic Renyi entropy of the dataset [18]. Using KECA for a data transformation often induces an angular structure in the transformed set. A spectral clustering method utilizing this particular structure is given in Section 4.4.

Chapter 5 concludes the theory part of this thesis by summarizing MSSC presented in [28]. The first sections explains the two-stage approach in more detail, including how the partition affinity matrix can be constructed by using the Cauchy-Schwarz divergence to utilize information from the entire dataset. Section 5.5 presents the novel changes in MSSC done in this thesis, followed by a discussion about parameter choices.

The experiments are reported in Chapter 6 and include the clustering of

some common benchmarks. These results are used to illustrate and discuss the important aspects of MSSC. As mentioned the focus is to give an understanding on how different parameter choices affects the result, rather than tweaking the parameters to find the optimal performance.

This thesis is concluded in Chapter 7. This chapter also includes a comprehensive list with brief points that can be considered for future work in order to expand MSSC.

1.2 Notation

It should come as no surprise that the clustering literature, like most if not all scientific fields, contains about as many notational conventions as there are published authors in the field. While this thesis includes information from many different articles and books, care has been taken to remain consistent throughout the chapters. The following general mathematical notation applies:

1. Small bold letters denotes vectors: $\mathbf{a}$, $\mathbf{b}$, $\boldsymbol{\mu}$. All vectors are column vectors unless explicitly stated otherwise.

2. Capital bold letters denotes matrices: $\mathbf{A}$, $\mathbf{B}$, $\boldsymbol{\Sigma}$.

3. The superscript T is used for the transpose operator: $\mathbf{A}^T$, $\mathbf{b}^T$.

4. All variables are real.

Some important variables, vectors and matrices can be found in the following table.

Notation	Dimension	Description
$\mathbf{x}$	$l \times 1$	stochastic vector or its realization (then often with a subscript)
$\mathbb{X}$		dataset of data vectors
$\mathbf{X}$	$n \times l$	dataset organized in a matrix with observations as rows
$\mathbf{y}$	$s \times 1$	transformed data vector
$\mathbb{Y}$		transformed dataset
$\mathbf{Y}$	$n \times s$	dataset after transformation organized in a matrix
$\boldsymbol{\mu}$	$l \times 1$	expectation or mean vector, $\boldsymbol{\mu} = \mathrm{E}\{\mathbf{x}\}$
K		kernel function
$\mathbf{K}$	$n \times n$	kernel matrix
$K_{i,j}$	1	element i, j of kernel matrix, $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$
$\mathbf{e}$	$n \times 1$	eigenvector of $\mathbf{K}$
$\mathbf{E}$	$n \times n$	matrix with eigenvectors $\mathbf{e}$ as columns
λ	1	eigenvalue of $\mathbf{K}$
$\boldsymbol{\Lambda}$	$n \times n$	diagonal matrix of eigenvalues
f		probability density function (pdf) of continuous random variable or vector
f		probability mass function of discrete random variable or vector (same symbol as pdf)
H		entropy
V		information potential
$\mathbf{H}$	$l \times l$	full bandwidth matrix
h	1	bandwidth or kernel size parameter

Notice that the kernel matrix $\mathbf{K}$ has dimension $m \times m$ when constructed based the m partitions rather than the full data set of n feature vectors (see Chapter 5. This also influences the size of all vectors and matrices related to $\mathbf{K}$.

Several abbreviations are used throughout the thesis. They are usually written explicitly when used for the first time in a chapter. A few of the more common are listed below.

EM	Expectation Maximization
ITL	Information Theoretical Learning
KDE	Kernel Density Estimation
KECA	Kernel Entropy Component Analysis
KPCA	Kernel Principal Component Analysis
pdf	Probability Density Function
pmf	Probability Mass Function
RKHS	Reproducing Kernel Hilbert Space

Some terms have different names that are used interchangeably in this thesis, for instance Parzen windowing is another name for kernel density estimation. Another example is that the feature vectors $\mathbf{x}$ are also referred to as data points or data vectors.

Chapter 2

Some Clustering Procedures

2.1 Introduction

2.1.1 Types of Clustering Procedures

In [16], clustering procedures are broadly divided into two groups, *hierarchical* and *partitional*.

Hierarchical methods recursively finds nested clusters. The hierarchical group can be further subdivided into two groups. For agglomerative clustering, each data point starts in its own cluster and at each step the most similar pair of clusters is merged. The other algorithm is divisive clustering; all points start in the same cluster which is recursively divided into smaller clusters.

Partitional methods do not impose a hierarchical structure, and once "the clustering rules" for the dataset have been found, it is divided into clusters according to these rules. In some clustering methods the rules are found by iterative procedures. The cluster assignment of a data point may change after each iteration, until the algorithm finds an assignment that is optimal in some sense. The k-means algorithm presented in Section 2.2 and Expectation Maximization for Gaussian mixture models in Section 2.3 are two examples of such iterative procedures.

Some partinional procedures use the proximity measures between the data points rather than the data points themselves. The proximity measures are organized in a matrix which is called the similarity matrix. These are called *spectral clustering* methods because they make use of the spectrum (eigenvalues and the corresponding eigenvectors) of the similarity matrix in some way. Kernel Principal Component Analysis (Section 2.5) and Laplacian eigenmaps (Section 2.6) are two spectral clustering methods.

A further distinction, between *clustering methods* and *clustering algo-*

rithms, is presented in [16]. A clustering method is a general strategy to find a solution to a clustering problem, while an algorithm is a realization of the clustering strategy defined by the method. K-means, which is presented in Section 2.2 is an algorithm, whereas Kernel Principal Component Analysis spectral clustering in Section 2.5 is a clustering method where the strategy is to find a new data representation by a non-linear transformation before clustering.

2.1.2 Methods and Algorithms Presented

Only *partitional* clustering methods are presented in this thesis as the methods used for Mean Shift Spectral Clustering (MSSC) are partitional. The first three chapters are about non-spectral clustering algorithms, with Section 2.2 covering the well known k-means algorithm. Section 2.3 describes clustering by using a Gaussian mixture model for the data and estimating the model parameters by Expectation Maximization (EM). The k-means can be seen as a special case of this method.

The mean shift algorithm and its application to clustering is presented in Section 2.4. It differs from k-means and EM for Gaussian mixtures in that it is a non-parametric method. The mean shift algorithm does not require the number of clusters as an input parameter. While it is not a spectral clustering method, it does have something in common with Kernel Principal Component Analysis (KPCA) in that it uses kernels.

Section 2.5 describes KPCA as a spectral clustering method. It uses kernels to perform a non-linear mapping (in mean shift, kernels are used for density estimation) which hopefully will ease the task of clustering. Laplacian eigenmaps in Section 2.6 is a graph based method which seeks to preserve the neighbourhood information. It is a spectral method with natural ties to clustering because it will provide a clustering solution if the graph is not fully connected.

Each section starts with an introduction which explains the reasoning behind the particular clustering approach. In Sections 2.2, 2.4 and 2.6, a special case of the general method is derived first. This will provide insight into when to use a particular clustering method as it shows cases where the algorithm is optimal in some sense. Because, as pointed out in [16], *"there is no best clustering algorithm"*.

There are also some plots of toy data examples for each section. These are meant to illustrate some aspects of each method, but are not meant to accurately summarize the overall performance of a method for real world applications.

An important aspect of all supervised and unsupervised learning is *data*

representation (choice of features), which influences the performance of all clustering algorithms [16]. Very little attention is given to the data representation (feature vectors) in this chapter, though any clustering method will fail to give a meaningful grouping of the data if the data is poorly represented.

2.2 The K-Means Algorithm

The k-means algorithm is one of the best known clustering algorithms [16]. Since it has relatively low computational complexity it is widely used, particularly for large data sets. K-means is an iterative clustering algorithm where the number of clusters, k, is an input parameter. It uses a crisp (hard) membership function, which means that each data sample is assigned to one and only one cluster. This leads to a non-differentiable cost function. Hence optimization is divided into two steps, where first the cluster representatives and then the memberships are kept constant while the other is optimized.

Each cluster is represented by a point-representative, which causes the algorithm to favour dense clusters. The cost function is a function of the memberships and the dissimilarity measure. The k-means algorithm can be derived from minimizing a cost function if the dissimilarity measure is the squared Euclidean distance. Then it can easily be shown that the point representatives that minimize the cost function are the k cluster means. Hence the name k-means.

The cluster memberships of the data samples are not known in advance and the initial cluster representatives are usually initialized at random. However, the number of clusters are assumed to be know. In the first step, each data sample is assigned to its closest cluster representative. Then new cluster representative for a particular cluster is set equal to the mean of the data samples assigned to that cluster. These steps are repeated to some convergence criteria are met.

2.2.1 Generalized Hard Algorithm Scheme

This section is based on the definitions in [39, Chapter 14.5]. Generalized Hard Algorithm Scheme (GHAS) is a family of clustering algorithms that are based around the same principles; hard membership functions and dissimilarity based cost function optimization. We will now look at what exactly this involves and use it to derive the GHAS.

The Membership Vector

We now wish to perform clustering on a dataset $\mathbb{X}$ consisting of n data (feature) vectors $\mathbf{x}_i$, $i = 1, 2, \ldots, n$ of dimension $l \times 1$. We write the dataset as: $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$. Assume that each data vector belongs to one and only one cluster, and that the total number of clusters is k. We can then define a $1 \times k$ membership vector, $\mathbf{u}_i$, for each data vector $\mathbf{x}_i$, with elements

$$u_{ij} \in \{0, 1\} \ , \qquad j = 1, \ldots, k \tag{2.1}$$

where $u_{ij} = 1$ if and only if $\mathbf{x}_i$ is in cluster j. If $\mathbf{x}_i$ is not in cluster j, $u_{ij} = 0$, which gives

$$\sum_{j=1}^{k} u_{ij} = 1 \tag{2.2}$$

In clustering we do not have training data with known labels (cluster assignments) available. The membership vectors $\mathbf{u}_i$ must then be estimated from the data. A meaningful group assignment of data points is the goal of all clustering algorithms. Another common trait for all algorithms is that one has to make some assumptions. While some of these assumptions are clearly stated, for instance as we will see that for k-means we assume that there are k clusters, other assumptions are more vague.

By defining the membership vector we have already made an assumption; that the data vectors belong to a single cluster. This can be quite reasonable in some cases, whereas in other cases it might be better to allow a data vector to belong to multiple clusters, so-called fuzzy membership. For illustrative purposes let us consider two simple examples.

Suppose we have designed some automatic system that want to classify different animals into different species, then (counting cross breeds as distinct species) a hard membership function would make the most sense. An animal can not be 50% lion and 50% eagle. However, if we cluster a set of feature vectors derived from web pages it might be reasonable to allow web pages to be members of more than one cluster. For instance we might see that one cluster corresponds to news pages and another to entertainment web pages and get various web pages in between, ranging from newspapers prone to sarcasm with a "weak" membership in the entertainment cluster to "infotainment" pages with equal memberships in the two, to entertainment pages somewhat influenced by the news.

As mentioned we need to estimate the membership vector. There are several approaches to this estimation. A common choice is to specify a cost function that our membership assignment seeks to minimize. This then leads to an optimization problem.

The Cost Function

We define the cost function for GHAS to be

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} d(\mathbf{x}_i, \boldsymbol{\theta}_j) \tag{2.3}$$

where $d(\mathbf{x}_i, \boldsymbol{\theta}_j)$ is a dissimilarity measure between data vector $\mathbf{x}_i$ and cluster C_j parameterized by its associated parameter "vector" $\boldsymbol{\theta}_j$. Note that the parameter "vector" might contain a combination of scalars, vectors and matrices; for instance for a multivariate normal distribution it might contain the expectation vector and covariance matrix of the distribution. It might also contain so-called cluster representatives; one or more data vectors which in some way are representative for the cluster.

Ideally, a cost function gives a large value when we make incorrect assignments. In unsupervised learning the correct labels are unknown and in practice our choice of cost function will define what kind of clusters we are looking for.

The Iterative Algorithm

Let us fix the parameter vectors, $\boldsymbol{\theta}_j$, $j = 1, \ldots, k$. Since for each $\mathbf{x}_i$ only one u_{ij} is non-zero, it is easy to see that we minimize the cost function in Eq. (2.3) by assigning $\mathbf{x}_i$ to the closest cluster defined by the specified dissimilarity measure. Thus

$$u_{ip} = \begin{cases} 1, \text{ if } d(\mathbf{x}_i, \boldsymbol{\theta}_p) = \min_{j=1,\ldots,k} d(\mathbf{x}_i, \boldsymbol{\theta}_j) \\ 0, \text{ otherwise} \end{cases} \tag{2.4}$$

Now, having found a cluster assignment, we can use the points assigned to each cluster to estimate its associated parameter vector $\boldsymbol{\theta}_j$ by keeping the membership functions constant. We do this by minimizing the cost function with respect to each of the parameter vectors. Taking the derivative with respect to a particular parameter vector $\boldsymbol{\theta}_p$ gives

$$\begin{aligned} \frac{\partial}{\partial \boldsymbol{\theta}_p} J(\boldsymbol{\theta}, U) &= \frac{\partial}{\partial \boldsymbol{\theta}_p} \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} d(\mathbf{x}_i, \boldsymbol{\theta}_j) \\ &= \sum_{i=1}^{n} u_{ip} \frac{\partial d(\mathbf{x}_i, \boldsymbol{\theta}_p)}{\partial \boldsymbol{\theta}_p} \end{aligned}$$

equating this to $\mathbf{0}$ gives

$$\sum_{i=1}^{n} u_{ip} \frac{\partial d(\mathbf{x}_i, \boldsymbol{\theta}_p)}{\partial \boldsymbol{\theta}_p} = \mathbf{0} \qquad p = 1, ..., k \tag{2.5}$$

which can be further simplified by using the expression for u_{ip} from (2.1):

$$\sum_{\mathbf{x}_i \in C_p} \frac{\partial d(\mathbf{x}_i, \boldsymbol{\theta}_p)}{\partial \boldsymbol{\theta}_p} = \mathbf{0} \qquad p = 1, ..., k \tag{2.6}$$

The iteration procedure for GHAS can then be summarized:

1. Initialize the parameter vectors.

2. Assign data vectors to the nearest cluster.

3. Update the estimate of the parameter vector of each cluster based on the data assigned to it.

4. Repeat Steps 2 and 3 until some convergence criterion is met.

2.2.2 The K-Means Algorithm as a Special Case of GHAS

The k-means algorithm is a hard algorithmic clustering method and can be seen as a special case of Generalized Hard Algorithm Scheme (GHAS). Note that we will see later in Section 2.3.3 that k-means also can be seen as a special case of the Expectation-Maximization algorithm. Because of its intuitive approach and simplicity, finding special conditions under which an algorithm is equivalent to k-means can provide valuable insight into an otherwise complicated clustering algorithm.

Assumptions

As a special case of GHAS, k-means use a hard membership function as defined in Eq. (2.1) and Eq. (2.2). It also characterizes a cluster by a single point representative, thus the parameter vectors $\boldsymbol{\theta}_j$ is a data vector (assumed) belonging to cluster C_j. This also gives us the option of choosing the dissimilarity measure between point and cluster to be a distance measure between two points.

The k-means algorithm can be derived from the cost function in Eq. (2.3) if the dissimilarity measure is the *squared* Euclidean distance between the data vector $\mathbf{x}_i$ and cluster representative $\boldsymbol{\theta}_j$:

$$\begin{aligned} d(\mathbf{x}_i, \boldsymbol{\theta}_j) &= ||\mathbf{x}_i - \boldsymbol{\theta}_j||^2 \\ &= \sum_{p=1}^{l} [\mathbf{x}_i(p) - \boldsymbol{\theta}_j(p)]^2 \end{aligned}$$

where the $||\cdot||$ denotes the Euclidean norm. Some write the Euclidean norm as $||\cdot||_2$ to indicate the use of a L_2-norm, but here the subscript will be dropped to simplifying the notation. Hence $||\cdot|| = ||\cdot||_2$ and the subscript will be reserved for cases where we use other norms. In vector notation we can write this (dropping the subscripts on $\mathbf{x}$ and $\boldsymbol{\theta}$) as

$$||\mathbf{x} - \boldsymbol{\theta}||^2 = (\mathbf{x} - \boldsymbol{\theta})^T(\mathbf{x} - \boldsymbol{\theta}) \tag{2.7}$$

Inserting this into the cost function in Eq. (2.3) gives

$$J(\boldsymbol{\theta}, U) = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}||\mathbf{x}_i - \boldsymbol{\theta}_j||^2 = \sum_{i=1}^{n}\sum_{j=1}^{k} u_{ij}(\mathbf{x}_i - \boldsymbol{\theta}_j)^T(\mathbf{x}_i - \boldsymbol{\theta}_j) \tag{2.8}$$

Before we proceed to minimize this cost function, we will look at the initialization of the k-means algorithm.

Initialization

The name of the algorithm comes from the fact that we seek a solution involving k clusters. This is a weakness of k-means and several other clustering algorithms as we might not know in advance the number of clusters we are seeking. However, due to its low computational complexity, it is possible to run the algorithm several times for different values of k and evaluate the result.

One way of achieving this is to optimize the Aikaike Information Criterion (AIC) or the Bayesian Information Criterion (BIC). An algorithm that uses ACI or BIC and does not require k as an input parameter presented in [29] as *X-means*. This will provide a solution which seeks to optimize the cost function AND the number of clusters according to the criterion defined.

In general these strategies seeks to compensate for the fact that the overall cost decreases as the number of clusters increases. In the extreme case, when the number of clusters k is equal to the number of data points n, each point will be in its own one point cluster and the cost defined in Eq. (2.8) will be zero. We will not focus on this here and simply assume that we know the number of clusters k.

Having chosen a k we need to initialize the k cluster representatives $\boldsymbol{\theta}_j$. This is usually done by randomly drawing (without replacement) k of the data vectors. As k-means might converge to a local minima, it might be beneficial to re-run the algorithm with a different initialization.

The Algorithm

We now fix the (initial) cluster representatives and wish to minimize the cost function in Eq. (2.8) with respect to the cluster assignments u_{ij}. As mentioned for the GHAS, it is easy to see that this is done by assigning each data vector to its closest cluster representative. Then Eq. (2.4) can be written as

$$u_{ip} = \begin{cases} 1, \text{ if } ||\mathbf{x}_i - \boldsymbol{\theta}_p||^2 = \min_{j=1,\ldots,k} ||\mathbf{x}_i - \boldsymbol{\theta}_j||^2 \\ 0, \text{ otherwise} \end{cases} \tag{2.9}$$

We then fix the cluster assignments and optimize with respect to the cluster representatives. For a given cluster representative we do this by taking the derivative with respect to $\boldsymbol{\theta}_p$ and equating this to zero:

$$\begin{aligned} \frac{\partial J(\boldsymbol{\theta}, U)}{\partial \boldsymbol{\theta}_p} &= \mathbf{0} \\ \frac{\partial}{\partial \boldsymbol{\theta}_p} \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} (\mathbf{x}_i - \boldsymbol{\theta}_j)^T (\mathbf{x}_i - \boldsymbol{\theta}_j) &= \mathbf{0} \\ \frac{\partial}{\partial \boldsymbol{\theta}_p} \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} \left(\mathbf{x}_i^T \mathbf{x}_i - 2\mathbf{x}_i^T \boldsymbol{\theta}_j + \boldsymbol{\theta}_j^T \boldsymbol{\theta}_j \right) &= \mathbf{0} \\ \sum_{i=1}^{n} u_{ip} 2(\mathbf{x}_i - \boldsymbol{\theta}_p) &= \mathbf{0} \\ \sum_{\mathbf{x}_i \in C_p} 2(\mathbf{x}_i - \boldsymbol{\theta}_p) &= \mathbf{0} \\ \Rightarrow \sum_{\mathbf{x}_i \in C_p} \mathbf{x}_i = \sum_{\mathbf{x}_i \in C_p} \boldsymbol{\theta}_p & \\ \sum_{\mathbf{x}_i \in C_p} \mathbf{x}_i = \boldsymbol{\theta}_p \sum_{\mathbf{x}_i \in C_p} 1 & \end{aligned}$$

By denoting the number of data vectors in cluster C_p as n_p and using that $n_p = \sum_{\mathbf{x}_i \in C_p} 1$, we get the final expression for updating the cluster representatives

$$\boldsymbol{\theta}_p = \frac{1}{n_p} \sum_{\mathbf{x}_i \in C_p} \mathbf{x}_i \tag{2.10}$$

We recognize this as the mean of the data vectors assigned to cluster C_p. This is done for each k cluster representatives, and hence the name k-means.

Here we started out with an assumption of a hard membership function and k clusters that we wanted to represent (parametrize) by a single point

and proceed to optimize this by minimizing the squared Euclidean distance between the vectors assigned to a cluster. However, other dissimilarity measures than the squared Euclidean distance could be used. The core of the algorithm is to assign points to the closest cluster representative and then finding new cluster representatives as the mean of the assigned points.

The iterative procedure can then be summarized by Eq. (2.9) and Eq. (2.10):

1. Initialize the cluster representatives $\boldsymbol{\theta}_j$, $j = 1, \ldots, k$. This can be done by drawing without replacement from the data vectors in feature space $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$.

2. Assign data vectors to the closest cluster C. Using the definition in Equation (2.9) will minimize the cost defined in Eq. 2.8.

3. For each cluster C_j , $j = 1, \ldots, k$ calculate a new cluster representative $\boldsymbol{\theta}_j$ as the mean of the points assigned to C_j in the previous step as defined in Equation (2.10).

4. Repeat steps 2 and 3 until no data vectors changes clusters.

It is possible to have some other convergence criterion in step 4, for instance based on the cost function, but this solution is sub-optimal compared to letting it run until no points change cluster assignments.

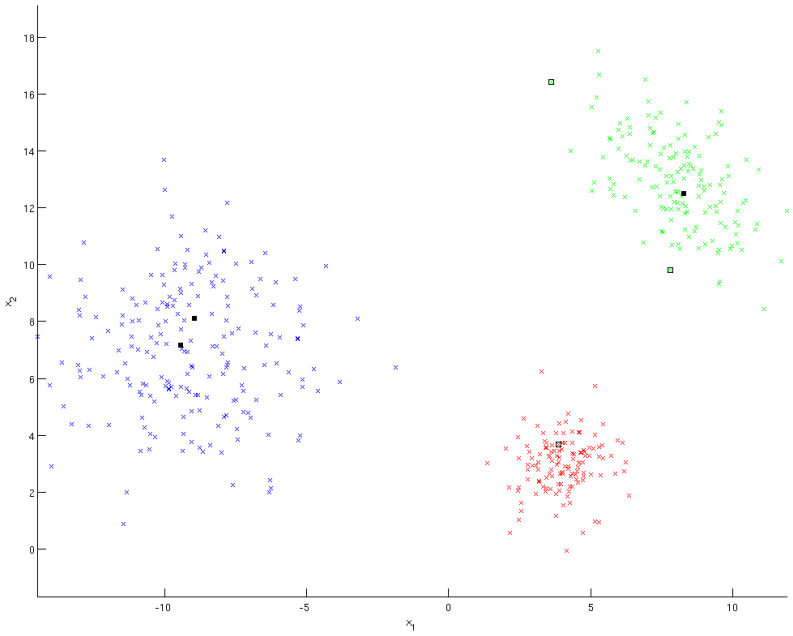


Figure 2.1: Dataset with two different initial sets of cluster representatives.

Figure 2.1 shows a toy dataset composed of three two dimensional multivariate normal distributions with different expectation vectors and covariance matrices. The data is plotted in three different colours according to which density they belong to.

Two different sets of initial cluster representatives are also shown in the figure, marked by black squares. In both cases the number of clusters provided to the algorithm was $k = 3$. We shall first look at the k-means algorithm with the empty black squares as the initial cluster representatives $\boldsymbol{\theta}$. Initially, two are located on the outskirts of the green cluster and one is in the middle of the red cluster.

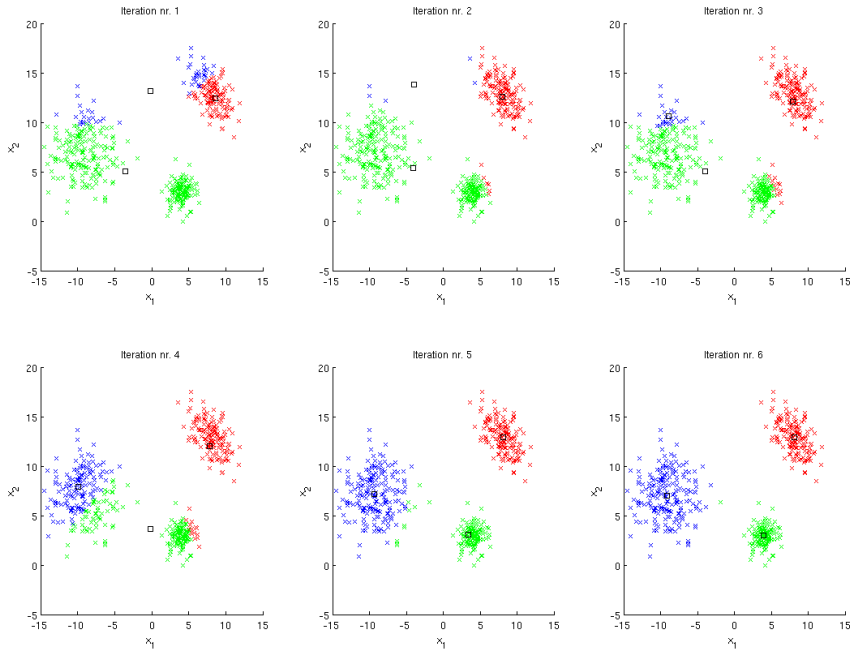


Figure 2.2: Convergence after 6 iterations.

Each subplot in Fig. 2.2 shows the clusters after an iteration. The black squares shows the location of each mean vector after each iteration. It should be apparent which cluster they belong to, even though they are not marked with the corresponding cluster colour. The algorithm converges after six iterations and the output is the clustering solution in the bottom right subplot.

The subplot illustrates nicely how the k-means algorithm is supposed to work. Even though the initial guess of staring points was wrong, the algorithm manages to provide a good clustering solution. If we look at Fig. 2.1, we see that though there are no initial points in the cluster on the left, the top and bottom cluster representatives each claim some of its point and start moving towards it. As the bottom cluster representative also represents the lowest cluster, it moves slower and is "pushed away" as the top representative claims the entire left cluster. The top right cluster is quite stable from the second iteration to convergence.

Note that the reason the colours in Fig. 2.2 does not match the colours in Fig. 2.1 is because the colours are based on which cluster the k-means algorithm label as 1, 2 and 3. This is done somewhat arbitrarily depending on the order and position of the initial guess of $\boldsymbol{\theta}$.

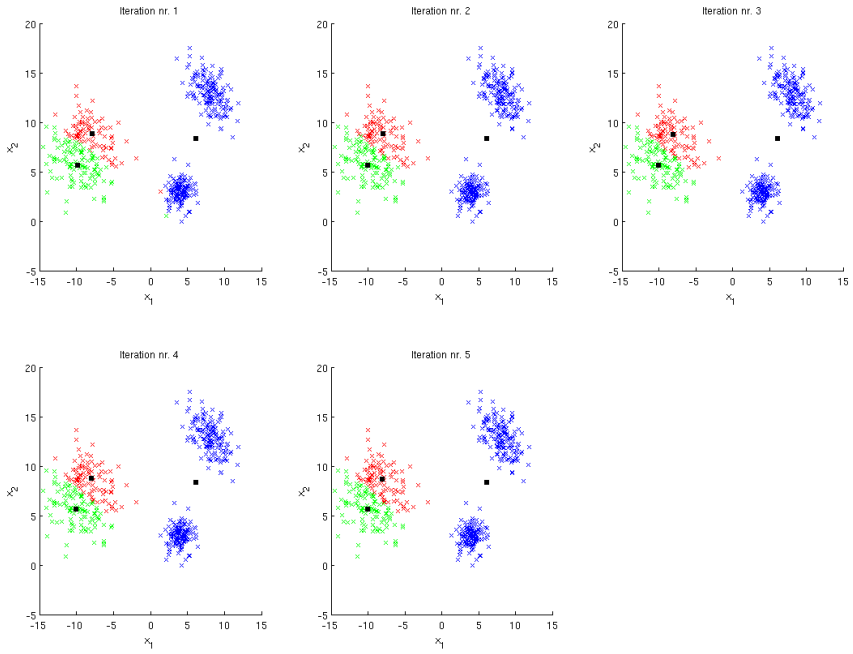


Figure 2.3: Convergance after 5 iterations.

Figure 2.3 shows the result of the k-means algorithm with the filled black squares in Fig. 2.1 as initial points; two located in the cluster on the left and one in the top right cluster. The algorithm converged after five iterations, where the criterion for convergence was that no change in cluster assignments should be made between iterations.

We see that the algorithm has converged to a local minimum. This possibility was mentioned in Section 2.2.2. Running the algorithm again with different initial cluster representatives might solve the problem if we select the solution with the lowest cost function.

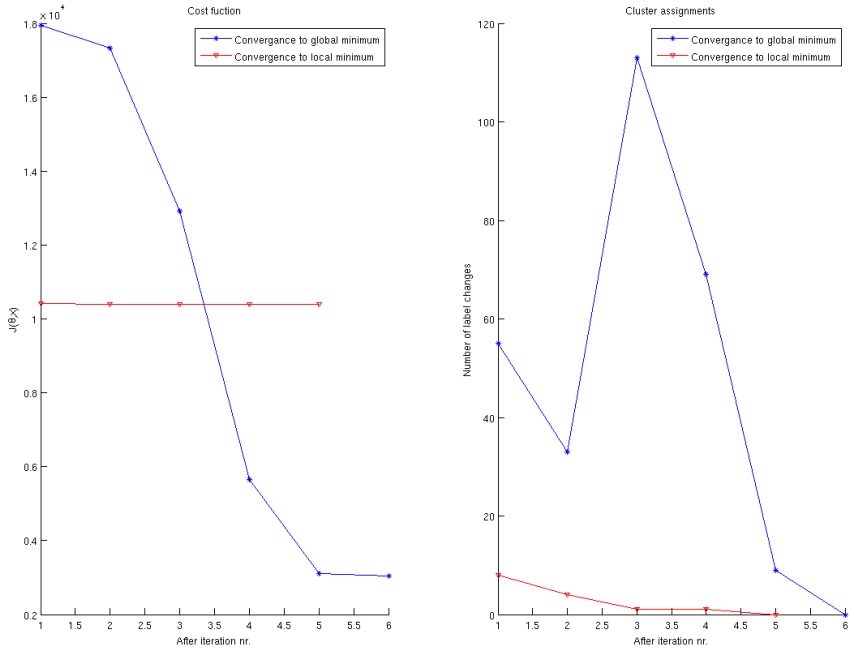


Figure 2.4: Cost function and number of cluster changes.

The left subplot of Fig. 2.4 shows the cost function (Eq. (2.8)) after each iteration for the iterative procedures shown in Fig. 2.2 (blue line) and Fig. 2.3 (red line). There is little change in the cost function corresponding to the iteration procedure show in Fig. 2.3, as would be expected since there is only minor changes in the cluster assignments.

This is shown in the right half of the figure which shows the number of data points that change clusters after each iteration, only a single point changes clusters after the third and fourth iteration. The first assignment of data vectors to the initial cluster representatives is not considered (as this would imply a change for all $n = 480$ data vectors).

The problem of convergence to local minima becomes more common as the clusters have significantly different sizes. This is one of the drawbacks of k-means. Some techniques for improving the algorithm can be found in [39, Chapter 14.5].

2.3 EM with Gaussian Mixture Models

This section is based on the description in [5, Chapter 9].

The EM-algorithm takes its name from the two steps preformed; the Expectation step and the Maximization step. The algorithm was originally developed to maximize difficult likelihoods by a sequence of maximization problems whose limit is the answer to the original problem. It is also used for to compensate for missing or censored data.

The algorithm is particularly suited for clustering in mixture model problems. Here the number and form of the distributions involved in the mixture model are assumed to be known. If the mixture components are Gaussian, the Expectation Maximization is particularly easy. This is because the multivariate normal distribution is only dependent on two parameters, the mean vector and the covariance matrix. In addition the maximum likelihood estimators for these are relatively simple to find.

After providing an initial guess for the parameters and probabilities of mixture components, the E and M steps are preformed iteratively until some convergence criteria are met. In the end, clustering is performed by assigning each data sample to the cluster which it has the highest probability of belonging to.

2.3.1 Mixture Models

General Mixture Models

With a mixture model for a stochastic vector $\mathbf{x}$, the probability density function (pdf), $f(\mathbf{x})$, can be modeled as a linear combination of density functions on the form

$$f(\mathbf{x}) = \sum_{j=1}^{J} f(\mathbf{x}|j)P_j \tag{2.11}$$

where P_j , $j = 1, \ldots, J$ are the mixture probabilities, that is the probability of belonging to component j. As it is a probability we have that

$$0 \leq P_j \leq 1 \qquad , \qquad j = 1, 2, \ldots, J \tag{2.12}$$

and a point can only belong to one mixture component, thus

$$\sum_{j=1}^{J} P_j = 1 \tag{2.13}$$

For the conditional probability density functions $f(\mathbf{x}|j)$ we have

$$\int_{\mathbf{x}} f(\mathbf{x}|j)d\mathbf{x} = 1 \tag{2.14}$$

From these conditions it should be apparent that the expression in Eq. (2.11) is a valid probability density function.

Given a sufficient of number mixture components J, any continuous density function can be modeled arbitrary closely [24].

If we have a mixture of parametric distributions, each of the J components will have associated parameters. These can be organized in J parameter "vectors", $\boldsymbol{\theta}_j$. As was mentioned in Section 2.2.1, these "vectors" might contain a combination of scalars, vectors and matrices. Equation (2.11) then becomes

$$f(\mathbf{x}) = \sum_{j=1}^{J} f(\mathbf{x}|j)P_j \tag{2.15}$$

Where conditioning on j implies that the distribution is parameterized by parameter vector $\boldsymbol{\theta}_j$.

Gaussian Mixture Models

A Gaussian mixture model is a special case of the general mixture model where every mixture component is a l-dimensional multivariate normal distribution. In other words

$$\mathbf{x}|j \sim \mathcal{N}_l(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \tag{2.16}$$

where $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$ are the expectation vector and covariance matrix of component j. Thus Eq. (2.15) becomes

$$f(\mathbf{x}) = \sum_{j=1}^{J} \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)P_j \tag{2.17}$$

$$= \sum_{j=1}^{J} P_j \, (2\pi)^{-l/2}|\boldsymbol{\Sigma}_j|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_j)^T\boldsymbol{\Sigma}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)\right) \tag{2.18}$$

Before we proceed to estimate these parameters and describe the clustering algorithm, we look at Gaussian mixture models in terms of latent variables as this will prove to be a useful formulation.

Mixture in Terms of Latent Variables

We now introduce the latent stochastic variable $\mathbf{z}$, where $\mathbf{z}$ is a $J \times 1$ vector where the elements are either 0 or 1:

$$z_j \in \{0, 1\} \tag{2.19}$$

The vector denotes which one of the J mixture components a data point is "drawn from". Thus

$$\sum_{j=1}^{J} z_j = 1 \tag{2.20}$$

as the data point can only belong to one mixture component.

We now look at the joint density of $\mathbf{x}$ and $\mathbf{z}$:

$$f(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}|\mathbf{z}) f(\mathbf{z}) \tag{2.21}$$

Note that $f(\mathbf{z})$ is actually discrete, thus it is a probability mass function (pmf) and not a pdf. This means that $f(\mathbf{z})$ is the probability of the stochastic vector z taking a particular realization where the k'th element equals 1, $k \in [1, J]$, and all other elements are 0. Hence it is just the probability of an arbitrary data point belonging to the k'th mixture component, which is already defined to be P_k. Then it is easy to see that

$$f(\mathbf{z}) = \prod_{j=1}^{J} P_j^{z_j} \tag{2.22}$$

The second component of Eq. (2.21) is the conditional density of $\mathbf{x}$ given $\mathbf{z}$. If the k'th element of $\mathbf{z}$ is 1, then this is just a multivariate normal density with mean vector $\boldsymbol{\mu}_k$ and covariance matrix $\boldsymbol{\Sigma}_k$. Thus

$$f(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{J} \left[\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right]^{z_j} \tag{2.23}$$

We can now find the (marginal) density for $\mathbf{x}$, $f(\mathbf{x})$ by summing (2.21) over

$\mathbf{z}$:

$$\begin{aligned} f(\mathbf{x}) &= \sum_{\mathbf{z}} f(\mathbf{x}|\mathbf{z}) f(\mathbf{z}) \\ &= \sum_{\mathbf{z}} \left[\prod_{j=1}^{J} \left[\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \right]^{z_j} \prod_{j=1}^{J} P_j^{z_j} \right] \\ &= \sum_{\mathbf{z}} \left[\prod_{j=1}^{J} \left[\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) P_j \right]^{z_j} \right] \\ &= \sum_{j=1}^{J} \mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) P_j \end{aligned}$$

Which is the formula for the mixture normal from Eq. (2.17). It might seem like nothing was gained by introducing $\mathbf{z}$. However, in unsupervised learning settings such as clustering, we do not know which data vector belongs to which mixture component. In fact, we do not even know if the data actually comes form a mixture distribution, it is just something we use to model the data.

Thus we need to estimate the stochastic vector $\mathbf{z}$. For this clustering algorithm we will use Maximum Likelihood Estimation (MLE), and being able to work on the joint distribution $f(\mathbf{x}, \mathbf{z})$ simplifies calculations.

Another advantage is that using Bayes' theorem we can define the conditional probability of $\mathbf{z}$ given $\mathbf{x}$, here denoted as $\gamma(z_k)$

$$\gamma(z_k) \equiv P(z_k = 1|\mathbf{x}) = \frac{P(z_k = 1) P(\mathbf{x}|z_k = 1)}{\sum_{j=1}^{J} P(z_j = 1) P(\mathbf{x}|z_j = 1)} \tag{2.24}$$

$$= \frac{P_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{J} P_j \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{2.25}$$

Then P_k can be seen as the prior probability of $z_k = 1$ and $\gamma(z_k)$ as corresponding the posterior probability given the observation $\mathbf{x}$. Thus $\gamma(z_k)$ is the posterior probability of a data vector belonging to mixture component k. It can also be seen as the *responsibility* component k takes for "explaining" the observation $\mathbf{x}$ [5].

2.3.2 Estimating Gaussian Mixture Model Parameters

Maximum Likelihood Estimation

Given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ of $l \times 1$ data vectors, we construct the $n \times l$ matrix $\mathbf{X}$ with each data vector corresponding to a row (observation):

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \tag{2.26}$$

where we assume that the observations $\mathbf{X}$ comes from a known parametric distribution with parameter vector $\boldsymbol{\theta}$; $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$. We can then define the *likelihood function*

$$\mathtt{L}(\boldsymbol{\theta}|\mathbf{X}) = f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n|\boldsymbol{\theta}) \tag{2.27}$$

as a function of the parameter vector $\boldsymbol{\theta}$ for observations $\mathbf{X}$. Maximum Likelihood Estimation is a method for finding estimates for the parameters in $\boldsymbol{\theta}$. The objective is to find an estimate $\hat{\boldsymbol{\theta}}$ that maximizes Eq. (2.27):

$$\hat{\boldsymbol{\theta}} = \arg\left[\max_{\boldsymbol{\theta}} \mathtt{L}(\boldsymbol{\theta}|\mathbf{X})\right] \tag{2.28}$$

Often it is more convenient to work with the natural logarithm of the likelihood function [42, Chapter 9]. Because the logarithm is a monotonically increasing function, maximizing it is equivalent to maximizing the likelihood function. The log likelihood function is defined as

$$\mathtt{l}(\boldsymbol{\theta}|\mathbf{X}) = \ln\left[\mathtt{L}(\boldsymbol{\theta}|\mathbf{X})\right] \tag{2.29}$$

This is particularly useful when the parametric distribution $f(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ belongs to the exponential family. When Eq. (2.29) is continuously differentiable in $\boldsymbol{\theta}$, the estimate $\hat{\boldsymbol{\theta}}$ may be obtained by looking for the maximum by differentiating the log-likelihood function [33, Chapter 6].

For data vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_2$ drawn independently from the same multivariate normal distribution, we get the well known maximum likelihood estimators

$$\hat{\boldsymbol{\mu}}_{\text{MLE}} = \bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \tag{2.30}$$

where $\bar{\mathbf{x}}$ is called the *sample mean*. For covariance the MLE is

$$\hat{\boldsymbol{\Sigma}}_{\text{MLE}} = \mathbf{S}_n = \frac{1}{n}\sum_{i=1}^{n} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \tag{2.31}$$

where $\mathbf{S}_n$ is called the *sample covariance* matrix. The subscript is to separate it from the unbiased estimate $\mathbf{S}_{n-1} = \frac{n-1}{n}\mathbf{S}_n$. $\bar{\mathbf{x}}$ is an unbiased estimate for $\boldsymbol{\mu}$.

Let the vector $\mathbf{p}$ be the $J \times 1$ vector of mixture probabilities of the mixture components

$$\mathbf{p} = [P_1, P_2, \ldots, P_J]^T \tag{2.32}$$

Similarly, let $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ denote the set all component expectation vectors $\boldsymbol{\mu}_j$ and covariance matrices $\boldsymbol{\Sigma}_j$ respectivly, $j = 1, \ldots, J$. The log likelihood function of a mixture of l-dimensional multivariate normal distribution can then be written as

$$\mathtt{l}(\mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma}|\mathbf{X}) = \sum_{i=1}^{n} \ln \left\{ \sum_{j=1}^{J} P_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{2.33}$$

Maximizing Eq. (2.33) poses a significant problem. The single component case, which the MLE estimators in Eq. (2.30) and Eq. (2.31) were derived from, did not depend on $\mathbf{p}$ and did not have a sum inside the logarithm.

MLE for Gaussian Mixture Models

Setting the derivatives of the log likelihood in Eq. (2.33) with respect to the expectations $\boldsymbol{\mu}_k$ equal to zero we get

$$0 = \sum_{i=1}^{n} \frac{P_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{J} P_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) \tag{2.34}$$

We recognize the first term as $\gamma(z_k)$ from Eq. (2.25) for a given $\mathbf{x}_i$. Defining n_k as

$$n_k = \sum_{i=1}^{n} \gamma(z_{ik}) \tag{2.35}$$

which we interpret at the effective number of points assigned to cluster C_k. We can now rewrite Eq. (2.34)

$$0 = \sum_{i=1}^{n} \gamma(z_{ik}) \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) \tag{2.36}$$

By multiplying by $\boldsymbol{\Sigma}_k$, which is assumed to be nonsingular, and rearranging we get

$$\hat{\boldsymbol{\mu}}_k = \frac{1}{n_k} \sum_{i=1}^{n} \gamma(z_{ik}) \mathbf{x}_i \tag{2.37}$$

which is a weighted mean, with the weight function being the posterior probability of a given data point belonging to the component. This seems reasonable, the probability of a point belonging to a particular mixture component determines how much we weight its contribution.

By similar reasoning it is possible to obtain MLE estimates of the covariance matrices $\mathbf{\Sigma}_k$

$$\hat{\mathbf{\Sigma}}_k = \frac{1}{n_k} \sum_{i=1}^{n} \gamma(z_{ik})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_k)^T \tag{2.38}$$

and, by including Eq. (2.13) as a constraint and using Lagrange optimization, prior probabilities P_k

$$\hat{P}_k = \frac{n_k}{n} \tag{2.39}$$

which is the ratio of the effective number of points in cluster C_k divided by the total number of points in the dataset.

All these estimators seems reasonable and appears to correspond well with the MLE for the single component case in Eq. (2.30) and Eq. (2.31). However, these results do not constitute a closed-form solution for the mixture model parameters as the responsibilities $\gamma(z_{ik})$ depends on those same parameters in a complicated way given by Eq. (2.25).

Instead we will use an iterative scheme for finding a solution to the MLE problem. This will be the Expectation Maximization algorithm.

The EM Algorithm

The Expectation Maximization algorithm takes its name from the two steps preformed. For estimating the Gaussian mixture parameters, the expectation step consists of replacing the responsibilities $\gamma(z_{ik})$ by their expected value based on the current estimate of parameters. Then the new parameters $(\boldsymbol{\mu}_k, \mathbf{\Sigma}_k, P_k)$ can be estimated by maximizing the likelihood function with respect to the different parameters.

The EM algorithm (where we drop the hat on $\hat{\boldsymbol{\mu}}_k, \hat{\mathbf{\Sigma}}_k, \hat{P}_k$ for notational convenience) can then be summarized as :

1. **Initialization.** Initialize the means $\boldsymbol{\mu}_k$, covariances $\mathbf{\Sigma}_k$ and mixing coefficients P_k for $k = 1, \ldots, J$.

2. **E step.** Evaluate the responsibilities based on the values of the current parameter estimates

$$\gamma(z_{ik}) = \frac{P_k \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{J} P_j \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_j, \mathbf{\Sigma}_j)}$$

for $k = 1, \ldots, J$.

3. M step. Re-estimate the parameters using the current current responsibilities

$$\begin{aligned}
\boldsymbol{\mu}_k^{\text{new}} &= \frac{1}{n_k} \sum_{i=1}^{n} \gamma(z_{ik}) \mathbf{x}_i \\
\boldsymbol{\Sigma}_k^{\text{new}} &= \frac{1}{n_k} \sum_{i=1}^{n} \gamma(z_{ik}) (\mathbf{x}_i - \boldsymbol{\mu}_k^{\text{new}})(\mathbf{x}_i - \boldsymbol{\mu}_k^{\text{new}})^T \\
P_k^{\text{new}} &= \frac{n_k}{n}
\end{aligned}$$

where

$$n_k = \sum_{i=1}^{n} \gamma(z_{ik})$$

for $k = 1, \ldots, J$.

4. Check for convergence. This can be done based on the log likelihood function

$$\mathtt{l}(\mathbf{p}, \boldsymbol{\mu}, \boldsymbol{\Sigma} | \mathbf{X}) = \sum_{i=1}^{n} \ln \left\{ \sum_{j=1}^{J} P_j \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

or on the convergence of the parameters. If the convergence criterion is not satisfied return to Step 2.

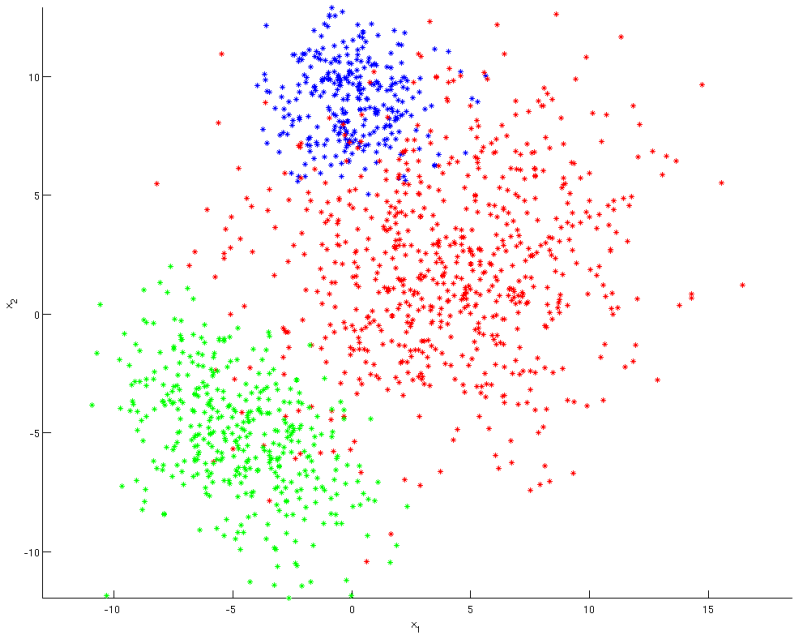


Figure 2.5: Dataset with three normal components.

Figure 2.5 shows a dataset generated by three multivariate normal components. In this example the data from the different components (marked with different colours) are overlapping and thus this presents a challenging estimation problem.

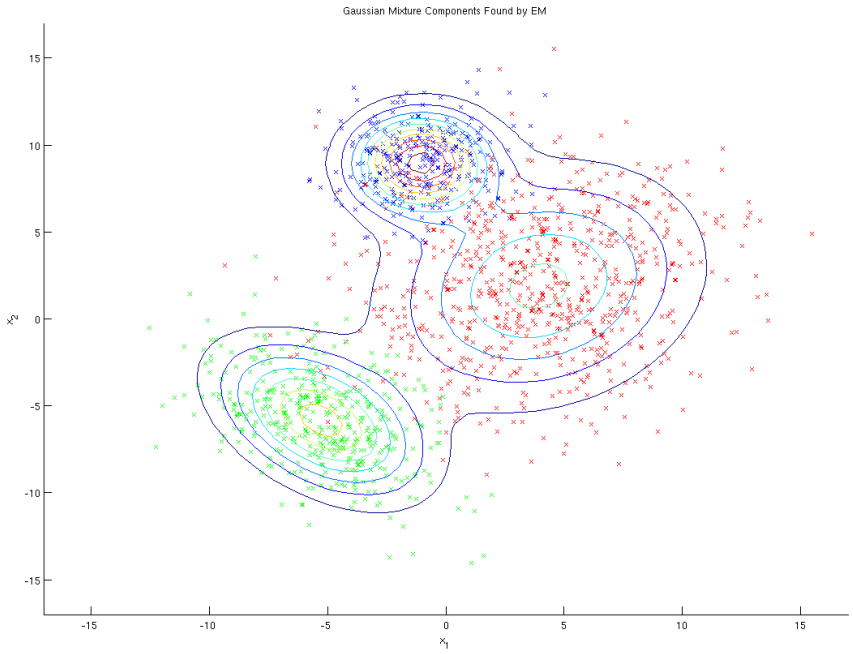


Figure 2.6: Gaussian mixtures fitted by EM.

However, as shown in Fig. 2.6, the EM algorithm manages to provide a reasonable good representation of the components. This is partly because the components had different covariance structure; blue had no correlation and medium variance, green had a large negative correlation and red had a large overall variance and a small positive correlation. Convergence took 93 iterations.

2.3.3 Clustering by EM of Mixture Parameters

Cluster Assignment

Select the number of clusters J, this will be the number of mixture components.

We now wish to assign each $l \times 1$ data vector in the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ to the mixture component which maximizes the posterior probability of the data vector belonging to that component. That is assign for each $\mathbf{x}_i$, $i = 1, \ldots, n$ to cluster C_m when

$$\gamma(z_{im}) = \max_{k=1,...,J} \gamma(z_{ik}) \tag{2.40}$$

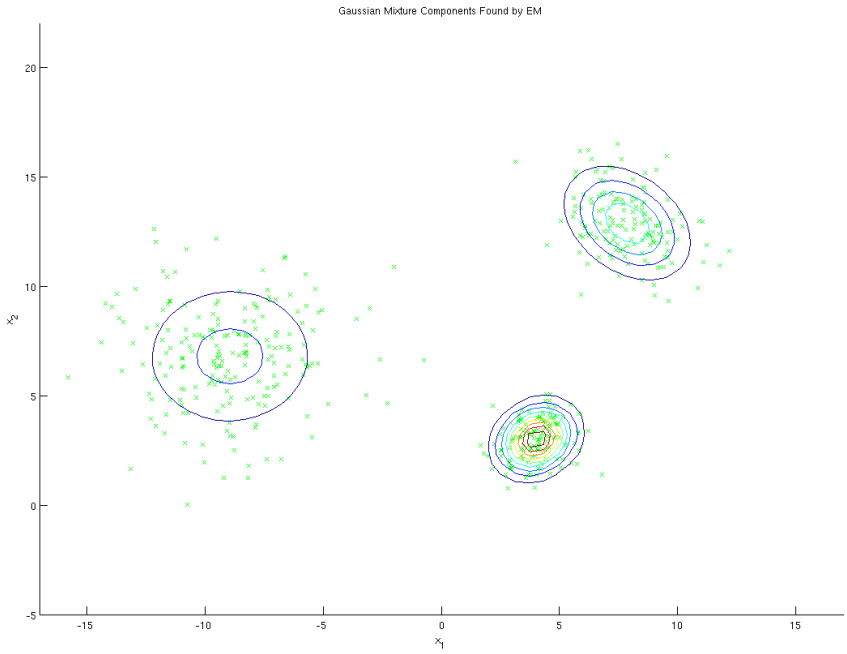


Figure 2.7: Dataset with mixture contours.

where the responsibilities $\gamma(z_{ik})$ are defined in Eq. (2.25).

Figure 2.7 shows a dataset generated by a combination of three multivariate normal (Gaussian) densities with the same parameters used in Fig. 2.1. The EM algorithm used to fit three Gaussian mixture components to the data and the figure shows the contour of each of these.

The three mean vectors were initializes as points drawn randomly from the data (just like k-means) and the covariance matrices were initialize as diagonal with the sample covariance (Eq. 2.31) of each variable (column of $\mathbf{X}$) as the diagonal elements.

Convergence was reached after 10 iterations. Assigning the points to the component with the highest posterior probability would correctly classify this dataset. Even though some of the points of the high variance cluster on the left is quite close to the bottom cluster, this cluster is correctly estimated to be very dense. The mixture probabilities and mean vectors returned by the algorithm is the same as those calculated by using the real cluster labels.

Relationship with K-Means

Whereas k-means uses a hard (crisp) membership function, clustering with the EM algorithm corresponds to a fuzzy membership functions because $\gamma(z_{ik}) \in [0, 1]$. Another difference is that the k-means algorithm does not consider the covariance matrices.

However, it is possible to derive the k-means as a special case of EM for Gaussian mixtures. Consider a Gaussian mixture model where the covariance matrices are given by

$$\mathbf{\Sigma}_k = \epsilon \mathbf{I} \qquad , \qquad k = 1, 2, \ldots, J \tag{2.41}$$

where $\mathbf{I}$ is the $l \times l$ identity matrix. These are not updated in the M-step.

Inserting this into Eq. (2.25) and using simple algebra gives that the posterior probability for a particular data point $\mathbf{x}_i$ is given by

$$\gamma(z_{ik}) = \frac{P_k \exp\left\{||\mathbf{x}_i - \boldsymbol{\mu}_k||^2/2\epsilon\right\}}{\sum_{j=1}^{J} P_j \exp\left\{||\mathbf{x}_i - \boldsymbol{\mu}_j||^2/2\epsilon\right\}} \tag{2.42}$$

If we now consider the limit $\epsilon \to 0$, we see that the term where $||\mathbf{x}_m - \boldsymbol{\mu}_j||^2$ is smallest will go to zero most slowly. Hence all responsibilities $\gamma(z_{ik})$ for $\mathbf{x}_i$ will go to zero, except for $\gamma(z_{im})$ which goes to unity.

This implies that each point $\mathbf{x}_i$, $i = 1, \ldots, n$ is assigned to its closest mean vector. Also, the estimate for $\boldsymbol{\mu}_k$ in Eq. (2.37) will then be the sum over all points assigned to cluster C_k and n_k will be exactly the number of points in this cluster.

Thus EM for Gaussian mixtures will give the k-means algorithm.

2.4 The Mean Shift Algorithm

The Mean Shift algorithm is a non-parametric, iterative mode seeking algorithm. It was originally presented in 1975 by Fukunaga and Hostetler [12]. It was significantly improved by Cheng in 1995 [6] which rekindled the interest for it. In recent years it has become popular in computer vision applications [7] and it will also be used for the first stage of the two-stage Mean Shift Spectral Clustering approach presented in Chapter 5.

We assume that the data in the feature space is sampled from an unknown probability density function (pdf) which we estimate. The procedure seeks out the local modes of the distribution, which is calculated by Kernel Density Estimation (KDE), also known as Parzen windowing. This is done by initializing a number of "mode-finding" vectors in the feature space and then moving them towards the local mean.

In practice these "mode-finding" vectors are usually initialized to be the data set in the feature space. Then the algorithm will associate each sample in the feature space with a local mode. Thus it is well suited for clustering; simply assign the data vectors that converged to the same mode to the same cluster. This means that the number of clusters is determined by the algorithm itself. Also, the probability density function is estimated using non-parametric methods. The fact that we do not have to make any assumptions about the distribution of the data or the number of clusters makes mean shift particularly attractive in the field of unsupervised learning.

However, one needs to specify the bandwidth parameters for the KDE. These parameters will significantly influence the output of the clustering, as the bandwidth decreases, the basin of attraction for each point will shrink, and when it goes to zero the algorithm will not move the mode-finding vectors from the initial position of the data vectors. Hence the output of a clustering procedure will be a one point cluster for each data vector. In the opposite case, when the bandwidth goes to infinity, the output will be one cluster located at the mean of the data vectors.

2.4.1 Kernel Density Estimation

Estimation of Probability Density Functions

Estimation of unknown probability density functions is a challenging task that often appears in connection with clustering. There are several approaches to this depending on how much we know about the data. In some cases we might now that the data comes from a certain type of parametric distribution (e.g., Uniform, Gaussian), but some or all parameters are unknown. Then the task is to estimate parameters from the data, for instance by using Maximum Likelihood Estimation (MLE).

This is the case for the Expectation Maximization algorithm, where the distribution is assumed to be a mixture of Gaussian distributions (Section 2.3.1). The parameters were then estimated by Expectation Maximization in Section 2.3.2.

Other times we might know something about the moments of a distribution, like the expectation (vector) and the (co-) variance (matrix). Then one would have to find the parametric distribution that best fits the data. However, in a clustering setting, such a priori information is rarely available. Hence we either have to make some assumptions or simplifications, or we can use non-parametric methods, where the only assumption is that the data is generated by some probability distribution equipped with a probability density function.

Histogram Methods

From basic statistics we know that the pdf can be approximated by a histogram. In the one-dimensional case, one can divide the x-axis into successive bins and count the number of data points (observations) that belong to each bin. By dividing by the total number of data points, thereby one gets an estimation of the probability of a data point belonging in that bin.

However, if the goal is to provide the estimate of the density in a particular point it is often better to place a bin so that the point was in its center. If we let n be the number of data points, $\mathbb{X} = \{x_1, x_2, \ldots, x_n\}$, and we define the bins to have a width h, we want to find an expression for the pdf in the point x. First we define the indicator function which, for a data point x_i , $i = 1, \ldots, n$, indicates if the data point is in a bin of width h centered at the test point x:

$$I_{x,h}(x_i) = \begin{cases} 1, \text{ if } |x_i - x| \leq \frac{h}{2} \\ 0, \text{ otherwise} \end{cases} \tag{2.43}$$

Then we can write the pdf estimate as

$$\hat{f}(x) = \frac{1}{h}\frac{1}{n}\sum_{i=1}^{n} I_{x,h}(x_i) \tag{2.44}$$

where $\sum_{i=1}^{n} I_{x,h}(x_i)$ is the number of points in the bin. Because of the discontinuous nature of the indicator function, the estimate will be discontinuous.

When the the data is multidimensional, the box of the histogram is replaced by a l-dimensional hypercube. The length of each side is h. Then the indicator function of the hypercube centered at $\mathbf{x}$ evaluated at the point $\mathbf{x}_i$ can be written as:

$$I_{\mathbf{x},h}(\mathbf{x}_i) = \begin{cases} 1, \text{ if } ||\mathbf{x}_i - \mathbf{x}||_\infty \leq \frac{h}{2} \\ 0, \text{ otherwise} \end{cases} \tag{2.45}$$

where $||\cdot||_\infty$ is the infinity norm which means that

$$||\mathbf{x}_i - \mathbf{x}||_\infty = \sup\{|x_i(1) - x(1)|, |x_i(2) - x(2)|, \ldots, |x_i(l) - x(l)|\} \tag{2.46}$$

Then we can "rephrase" Eq. (2.44) for the l-dimensional case:

$$\hat{f}(\mathbf{x}) = \frac{1}{h^l}\frac{1}{n}\sum_{i=1}^{n} I_{\mathbf{x},h}(\mathbf{x}_i) \tag{2.47}$$

The sum is still equal to the number of points that fall inside the hypercube. However, as with the one dimensional case, the pdf estimate will be discontinuous.

Parzen Windows

As presented in [39, Chapter 2.5.6], the idea is now to replace the discontinuous indicator function with a smooth function, $K(\mathbf{x}, \mathbf{x}_i)$. It can be shown that if

$$K(\mathbf{x}, \mathbf{x}_i) \geq 0 \tag{2.48}$$

and

$$\int_{\mathbf{x}} K(\mathbf{x}, \mathbf{x}_i) d\mathbf{x} = 1 \tag{2.49}$$

replacing the indicator function with $K(\mathbf{x}, \mathbf{x}_i)$ will give a valid pdf estimate. Such functions are called *Parzen windows* or *kernels*, which is why density estimation based on these functions are known as Kernel Density Estimation (KDE) or Parzen Windowing. The theory behind kernels is quite extensive and will not be covered in detail here. The relationship between different kernels is described in [6] and some further discussion on the choice of parameters of a kernel can be found in [7].

The estimate of the pdf at $\mathbf{x}$ given the data $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ and a kernel function K satisfying Eq. (2.48) and Eq. (2.49) can then be written as

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} K(\mathbf{x}, \mathbf{x}_i) \tag{2.50}$$

Note that the kernel function will have some associated parameters which is not specified here. Generally, different kernels will require different parameters.

It was noted in [28] that KDE may fail to provide accurate density estimation in high dimensional spaces. However, clustering is a simpler problem, and density estimates that are not acceptable for modeling the data can still achieve successful clustering.

The Gaussian Kernel

The Gaussian kernels are a typical choice in Parzen Windowing [39, Chapter 2.5.6]. Inserting the Gaussian kernel into Eq. (2.50) gives

$$\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} (2\pi)^{-l/2} |\mathbf{H}|^{-1/2} \exp\left(-\frac{1}{2} (\mathbf{x} - \mathbf{x}_i)^T \mathbf{H}^{-1} (\mathbf{x} - \mathbf{x}_i) \right) \tag{2.51}$$

where l is the dimension of the data and $\mathbf{H}$ is called the bandwidth matrix. It has the same role as the covariance matrix in a multivariate normal distribution, and hence it defines the orientation and size of the Gaussians used to estimate the density.

A fully parameterized bandwidth matrix will increase the complexity of estimation and in practice $\mathbf{H}$ is chosen to be either diagonal or proportional to the identity matrix $h^2\mathbf{I}$ [7]. The advantages should be clear as in the first case we would need to specify $\frac{l(l+1)}{2}$ parameters, whereas choosing a diagonal matrix reduces the number of parameters to l and in the last case only one parameter is needed, regardless of the dimension of the data.

Choosing something other than a single bandwidth parameter could be justifiable if we have reason to doubt the validity of the Euclidean metric for the feature space [7]. Unless otherwise noted we will assume that a single bandwidth parameter is sufficient.

$$\mathbf{H} = h^2\mathbf{I} \tag{2.52}$$

This parameter is also known as the *kernel size*.

Inserting this into Eq. (2.51) gives

$$\begin{aligned}\hat{f}(\mathbf{x}) &= \frac{1}{n}\sum_{i=1}^{n}(2\pi)^{-l/2}|(h^2\mathbf{I})|^{-1/2}\exp\left(-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^T(h^2\mathbf{I})^{-1}(\mathbf{x}-\mathbf{x}_i)\right)\\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{(2\pi)^{l/2}h^l}\exp\left(-\frac{(\mathbf{x}-\mathbf{x}_i)^T(\mathbf{x}-\mathbf{x}_i)}{2h^2}\right)\end{aligned}$$

Which can be simplified further by writing it as

$$\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{(2\pi)^{l/2}h^l}\exp\left(-\frac{1}{2}\left|\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|\right|^2\right) \tag{2.53}$$

where $||\cdot||$ is the Euclidean norm. Equation (2.53) also serves to illustrate why we might consider to specify further bandwidth parameters if we doubt the Euclidean metric is valid.

2.4.2 Mean Shift Algorithm Variations

Mean Shift with Gaussian Kernels

The Gaussian kernel is a popular choice for the Kernel Density Estimation performed in mean shift clustering [9]. In [31] it was shown that the choice of Gaussian kernels is connected with minimization of Renyi's entropy (see Section 3.4).

The intuition that mean shift is a form of gradient ascent, which was pointed out in [12], also makes Gaussian kernels attractive since it allows the mean shift algorithm to be defined from a gradient perspective. However, the

mean shift algorithm can be used even when the kernel is more complicated, for instance if the bandwidth matrix is fully parameterized. In such cases the mean shift algorithm might not be interpreted as gradient ascent.

None the less, for a Gaussian kernel with the bandwidth matrix defined in Eq. (2.52), we can derive the mean shift algorithm. Given n data vectors, $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ in the $\mathbb{R}^l$ feature space, the pdf is estimated by KDE with the Gaussian kernel function $K(\mathbf{x})$ defined in Eq. (2.53). We now take the gradient with respect to $\mathbf{x}$ of the estimate:

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{(2\pi)^{l/2}h^l} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) \frac{-2(\mathbf{x}-\mathbf{x}_i)}{2h^2} \tag{2.54}$$

$$= \frac{1}{n(2\pi)^{l/2}h^l} \sum_{i=1}^{n} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) \frac{(\mathbf{x}_i-\mathbf{x})}{h^2} \tag{2.55}$$

$$= \frac{1}{n(2\pi)^{l/2}h^{l+2}} \sum_{i=1}^{n} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) (\mathbf{x}_i-\mathbf{x}) \tag{2.56}$$

The local modes of the estimated density function are located among the zeros of the density gradient $\nabla \hat{f}(\mathbf{x}) = \mathbf{0}$. We can now rearrange this stationary point equation to obtain an iterative fixed point scheme

$$\frac{1}{n(2\pi)^{l/2}h^{l+2}} \sum_{i=1}^{n} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) (\mathbf{x}_i-\mathbf{x}) = \mathbf{0}$$

$$\Rightarrow \sum_{i=1}^{n} \mathbf{x} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right) = \sum_{i=1}^{n} \mathbf{x}_i \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right)$$

$$\Rightarrow \mathbf{x} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{||\mathbf{x}-\mathbf{x}_i||^2}{2h^2}\right)}$$

If we now denote the solution for the next iteration step, $\mathbf{x}^{(\tau+1)}$, as being given by a function of the current iteration step $m(\mathbf{x}^{(\tau)})$, we can write the algortihm as

$$\mathbf{x}^{(\tau+1)} = m(\mathbf{x}^{(\tau)}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i \exp\left(-\frac{||\mathbf{x}^{(\tau)}-\mathbf{x}_i||^2}{2h^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{||\mathbf{x}^{(\tau)}-\mathbf{x}_i||^2}{2h^2}\right)} \tag{2.57}$$

where $m(\mathbf{x})$ is the sample mean weighted by the kernel centered at $\mathbf{x}$. This is what was called a mode-finding vector. In other words, each data point is weighted by its contribution to the density estimate at the point $\mathbf{x}$. The term $m(\mathbf{x}) - \mathbf{x}$ was called "mean shift" in [12].

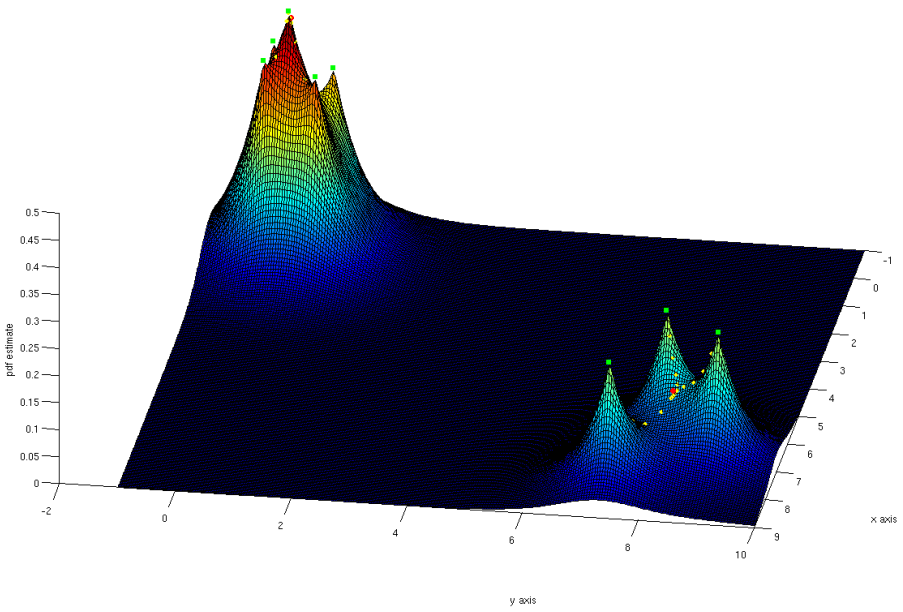


Figure 2.8: KDE with $h = 0.5$ and mean shift.

Figure 2.8 shows the pdf estimate using a Gaussian kernel with bandwidth $h = \frac{1}{2}$. The mean shift procedure has also been plotted in the same figure. The data points used are marked with green squares, the iteration steps with yellow diamonds and the final position of the mode-finding vectors after 20 iterations are marked with red squares. The height of all points are equal to the pdf estimate for that x & y coordinate pair. Note that the mean shift algorithm was set to run for 20 iterations regardless of convergence criterion and that a small constant ($\delta = 0.01$) was added to the height of the final "mode-finding" vectors to improve visibility. The algorithm was non-blurring as described in Section 2.4.2.

We notice that the algorithm has converged to a two cluster solution, one for the tight group of five points and one for the more widespread group of three points. In the first case the cluster is located near the data point which coincides with the mode (the global maximum). The second cluster lies in between three local modes, and appears to be located at a saddle point in the pdf estimate. This is undesirable if we were looking to find the position of local modes. However, it might be a good for clustering applications, particularly for cases such as this where a region contains few data points. This is because it might cause points that are located at local

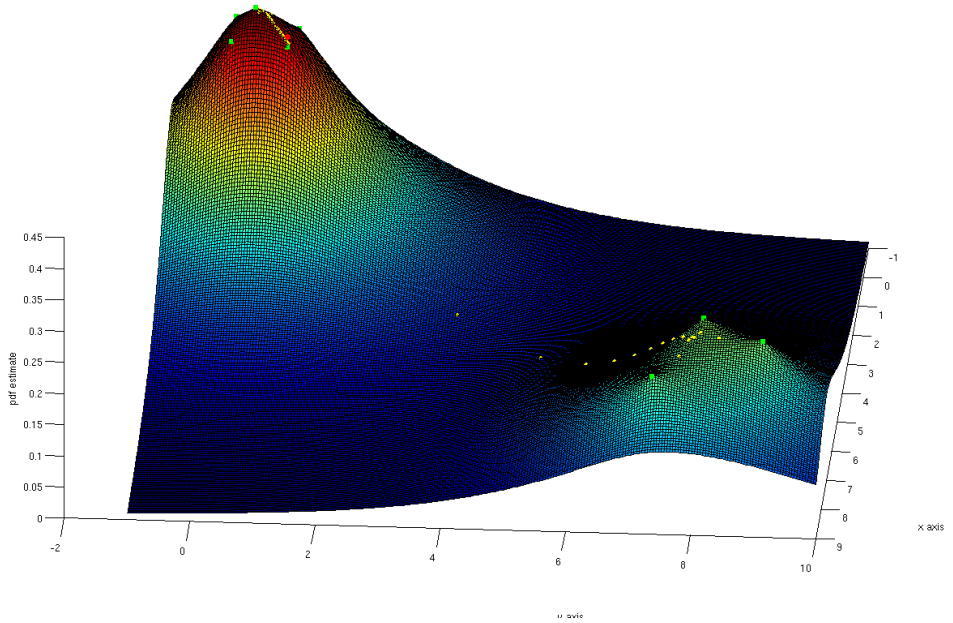


Figure 2.9: KDE with $h = 1$ and mean shift.

maxima (because the kernel size is to small relative to this region) to be clustered together. Otherwise the final clustering result would have a higher number of very small and one-point clusters. Note that if the kernel size is too small, this will be the case regardless.

Figure 2.9 illustrates how the choice h affects the Parzen Window estimate and the mean shift algorithm. The kernel size has doubled compared to the one used in Fig. 2.8 and the orientation of the plot view is different, but everything else remains the same. The pdf estimate looks much smoother, as expected, the larger h, the more smoothing. In the 20 iterations, the mode-finding vectors have converged to a single cluster, which is located closer to the three isolated points compared to Fig. 2.8. We notice that the mode-finding vectors move much further in the low density region between the two groups than close to the peak.

Generalized Mean Shift

We can generalize Eq. (2.57) to apply for a general kernel by replacing the expression for the Gaussian kernel by $K(\mathbf{x} - \mathbf{x}_i)$. Hence

$$\mathbf{x}^{(\tau+1)} = m(\mathbf{x}^{(\tau)}) = \frac{\sum_{i=1}^{n} \mathbf{x}_i K(\mathbf{x} - \mathbf{x}_i)}{\sum_{i=1}^{n} K(\mathbf{x} - \mathbf{x}_i)} \tag{2.58}$$

is the general mean shift algorithm. While the "gradient intuition" might not be as straight forward to show for a general kernel, the interpretation that each data point is weighted by its contribution to the density estimate by the kernel function K at the point $\mathbf{x}$ still applies.

So far we have focused on the finding the mean shift for an arbitrary point $\mathbf{x}$. For clustering applications it is intuitive to initialize the mode-finding vectors to be the data samples, $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$.

Blurred Mean Shift

In the original algorithm presented in [12], for every iteration, the pdf is then estimated again using the new, mean-shifted dataset. This can be seen as, for each iteration step, moving every data point towards a local mode of the current density estimate. The procedure is repeated until convergence. If we let the dataset be as defined in Section 2.4.2, the algorithm can be summarized as

1. Given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, initialize the dataset of "mode-finding" vectors for the iteration procedure as $\mathbb{X}^{(0)} = \left\{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, ..., \mathbf{x}_n^{(0)}\right\} = \mathbb{X}$

2. Find $\mathbb{X}^{(\tau+1)}$ by computing $\mathbf{x}_j^{(\tau+1)} = m(\mathbf{x}_j^{(\tau)}) = \frac{\sum_{i=1}^{n} K\left(\mathbf{x}_j^{(\tau)} - \mathbf{x}_i^{(\tau)}\right)\mathbf{x}_i^{(\tau)}}{\sum_{i=1}^{n} K\left(\mathbf{x}_j^{(\tau)} - \mathbf{x}_i^{(\tau)}\right)}$ for $j = 1, 2, ..., n$

3. Repeat step 2 until some convergence criterion is met.

4. Assign points that converged to the same local mode to the same cluster.

Note that since the mode-finding vectors are calculated based on the dataset of the previous iteration, they move towards the local modes in the KDE estimate based on the mode-finding vectors and not the original data. Hence,

while the algorithm will converge towards a local mode, it will *not* necessarily be a local mode in the original pdf estimate.

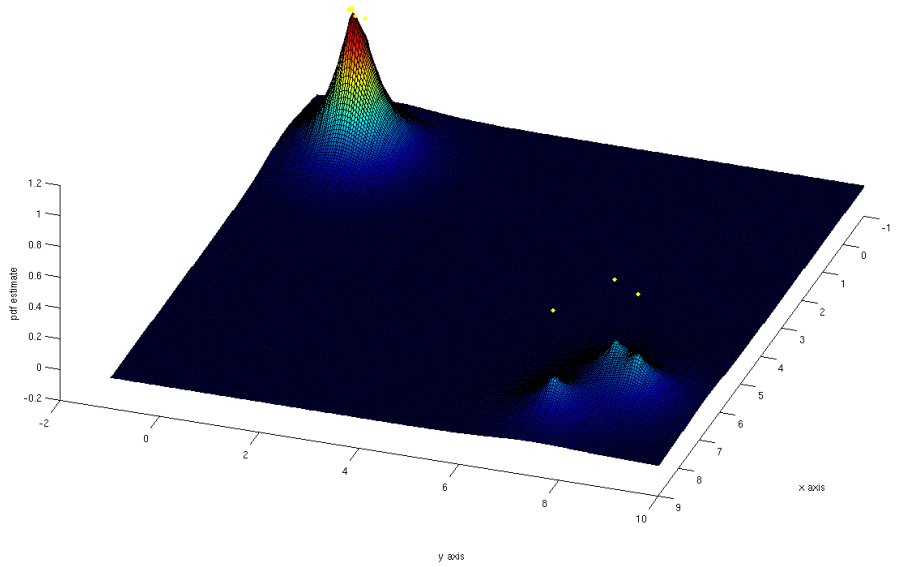


Figure 2.10: KDE with $h = 0.5$ after one "blurring".

This was first pointed out in [6] where it was referred to as "blurring". This convention was continued in [31]; *"As the new datasets are produced we forget the previous one which gives rise to the blurring process."*. Figure 2.10 shows the pdf estimate based on the "mode-finding" vectors (shown as yellow diamonds above the surface) after the first iteration. The kernel is the same as the one used in Fig. 2.8, $h = 0.5$, and when we compare the two figures we see the blurring effect.

It was showed in [31] that the solution found by blurred mean shift with a Gaussian kernel *"... minimizes Renyi's quadratic entropy of the dataset and hence is unstable by definition."*.

Non-Blurring Mean Shift

Cheng's article [6] pointed out the blurring effect caused by moving the data points and generalized the mean shift algorithm by proposing that the data set could be kept constant. The algorithm will be quite similar to the blurring one:

1. Given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, initialize the dataset of "mode-finding" vectors for the iteration procedure as $\mathbb{X}^{(0)} = \left\{\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, ..., \mathbf{x}_n^{(0)}\right\} = \mathbb{X}$

2. Find $\mathbb{X}^{(\tau+1)}$ by computing $\mathbf{x}_j^{(\tau+1)} = m(\mathbf{x}_j^{(\tau)}) = \frac{\sum_{i=1}^{n} K\left(\mathbf{x}_j^{(\tau)} - \mathbf{x}_i\right)\mathbf{x}_i}{\sum_{i=1}^{n} K\left(\mathbf{x}_j^{(\tau)} - \mathbf{x}_i\right)}$ for $j = 1, 2, ..., n$

3. Repeat step 2 until some convergence criterion is met.

4. Assign points that converged to the same local mode to the same cluster.

As with blurring mean shift, the Gaussian kernel is optimal in that it *"... minimizes Renyi's "cross" entropy where the local stationary solutions are modes of the dataset."* [31]. This is, contrary to the blurring case, a stable solution.

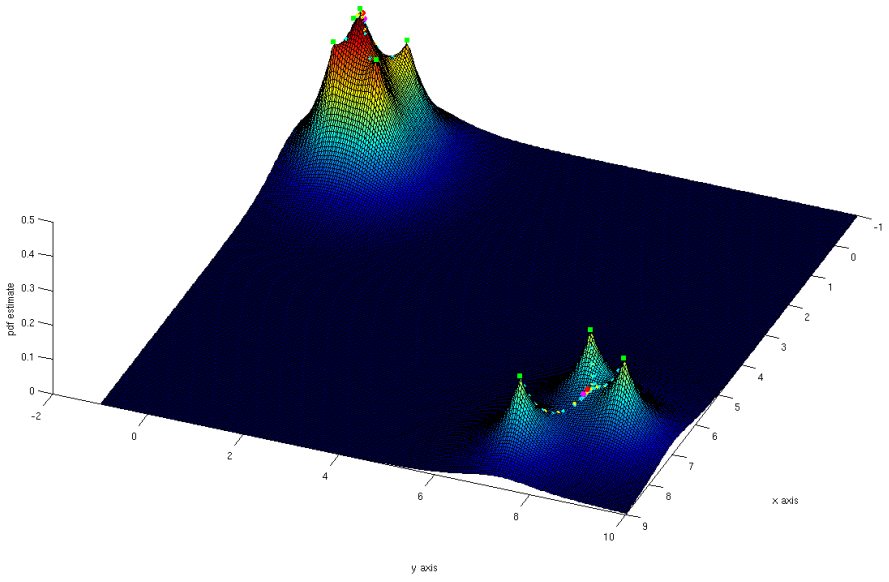


Figure 2.11: $h = 0.5$, blurring and non-blurring mean shift.

In general blurring and non-blurring mean shift will provide different results. Figure 2.11 is just Fig. 2.8 with the blurred mean shift iterations plotted with cyan diamonds and the result after 20 iterations is marked with a magenta circle. We see that the clustering will be the same, but the position of the final mode-finding vectors are slightly different. This toy data set is not well suited for illustrating the difference between the blurring and non-blurring solution, which generally can provide different clustering results. In some cases, such as this one, they will provide very similar results and blurring can be used to speed up convergence.

2.5 Kernel PCA Spectral Clustering

As already stated, there is no universally best clustering method. Which clustering method we choose should be decided by the problem at hand, which again depends on how we represent our data in terms of features. The algorithms discussed so far all have in common that they assign points around the dense regions in the feature space to the same cluster.

This will lead to poor performance when the center of mass is insufficient for describing the cluster. The clusters might not be separable in the feature space. To get good results in such cases we either need to make a complicated (ad hoc) clustering algorithm in the feature space, or we do a non-linear data transformation to a space where well known clustering algorithms can do the job.

Kernel Principal Component Analysis (KPCA) is a clustering *method* (as opposed to an *algorithm* according to the terminology used in [16]) that uses KPCA to do such a non-linear mapping. The strategy is to use the "kernel trick" to implicitly map the feature vectors to a higher dimensional space where hopefully the cluster structure of the data will be easily found by simple non-spectral clustering algorithms. KPCA can also be applied to both reconstruction and de-noising of data [25].

2.5.1 Principal Component Analysis

This section is based on [20, Chapter 8]. Regular Principal Component Analysis (PCA) is a data transformation method that projects the data onto a set of orthogonal directions. By choosing directions (axes) that captures most of the variance of the data, the dimension can be reduced with minimal quadratic error.

It seeks to explain the covariance structure of the *variables* of the dataset through a few linear combinations of these variables.

Assume that $\mathbf{x}$ is a stochastic vector with l random variables, $x_1, x_2, \ldots, x_l$. We now wish to find a linear combination of these variables that maximizes the variability. Geometrically this represents a rotation of the original system. We define the linear combinations as:

$$\begin{aligned} y_1 &= \mathbf{a}_1^T\mathbf{x} = a_{11}x_1 + a_{12}x_2 + \ldots + a_{1l}x_l \\ y_2 &= \mathbf{a}_2^T\mathbf{x} = a_{21}x_1 + a_{22}x_2 + \ldots + a_{2l}x_l \\ \vdots & \qquad\qquad \vdots \\ y_l &= \mathbf{a}_l^T\mathbf{x} = a_{l1}x_1 + a_{l2}x_2 + \ldots + a_{ll}x_l \end{aligned}$$

Where $\mathbf{a}_i$ are the vectors of scalar coefficients that produce the linear combinations y_i where $i = 1, \ldots, l$. Let $\mathbf{\Sigma}$ be the covariance matrix of $\mathbf{x}$. Then we can find the following expressions for y:

$$\text{Var}\{y_i\} = \mathbf{a}_i^T\mathbf{\Sigma}\mathbf{a}_i \qquad i = 1, 2, \ldots, l \tag{2.59}$$

$$\text{Cov}\{y_i, y_j\} = \mathbf{a}_i^T\mathbf{\Sigma}\mathbf{a}_j \qquad i, j = 1, 2, \ldots, l \tag{2.60}$$

We wish to find the vectors of scalar coefficients $\mathbf{a}_i$ that produce the linear combinations, y_i, with maximum variance (as defined in Eq. (2.59)) for $i = 1, \ldots, l$. From Eq. (2.59) we see that we need to impose some constraint on $\mathbf{a}$ as

$$\lim_{||\mathbf{a}||^2 \to \infty} \mathbf{a}^T\mathbf{\Sigma}\mathbf{a} = \infty \tag{2.61}$$

because the covariance matrix $\mathbf{\Sigma}$ is positive semi-definite (psd).

We also want the linear combinations y_i to be uncorrelated, otherwise if we had found that a single linear combination, $y_j = \mathbf{a}_j^T\mathbf{x}$, that maximizes the variance, we would have $y_j = y_1 = y_2 = \ldots = y_l$. The principal components are then defined as

The first principal component is the linear combination $\mathbf{a}_1^T\mathbf{x}$ that maximizes $\text{Var}\left\{\mathbf{a}_1^T\mathbf{x}\right\}$ subject to $\mathbf{a}_1^T\mathbf{a}_1 = 1$

The second principal component is the linear combination $\mathbf{a}_2^T\mathbf{x}$ that maximizes $\text{Var}\left\{\mathbf{a}_2^T\mathbf{x}\right\}$ subject to $\mathbf{a}_2^T\mathbf{a}_2 = 1$ and $\text{Cov}\left\{\mathbf{a}_1^T\mathbf{x}, \mathbf{a}_2^T\mathbf{x}\right\} = 0$

And so on with, the i'th component;

The i'th principal component is the linear combination $\mathbf{a}_i^T\mathbf{x}$ that maximizes $\text{Var}\left\{\mathbf{a}_i^T\mathbf{x}\right\}$ subject to $\mathbf{a}_2^T\mathbf{a}_2 = 1$ and $\text{Cov}\left\{\mathbf{a}_i^T\mathbf{x}, \mathbf{a}_j^T\mathbf{x}\right\} = 0$ for $j < i$

From this we can formulate the following optimization problem: Maximize

$$\mathbf{a}_i^T \boldsymbol{\Sigma} \mathbf{a}_i \qquad i = 1, 2, \ldots, l \tag{2.62}$$

with respect to $\mathbf{a}$, subject to the constraints

$$\begin{aligned} ||\mathbf{a}_i||^2 = \mathbf{a}_i^T \mathbf{a}_i &= 1 \qquad i = 1, 2, \ldots, l & (2.63) \\ \text{Cov}\{y_i, y_j\} = \mathbf{a}_i^T \boldsymbol{\Sigma} \mathbf{a}_j &= 0 \qquad i, j = 1, 2, \ldots, l & (2.64) \end{aligned}$$

Then it is possible to formulate a Lagrange optimization problem for a particular $\mathbf{a}$

$$L(\mathbf{a}, \lambda) = \mathbf{a}^T \boldsymbol{\Sigma} \mathbf{a} - \lambda \left(\mathbf{a}^T \mathbf{a} - 1 \right) \tag{2.65}$$

where λ is the Lagrange multiplier. The objective is now to maximize the Lagrange function. Taking the gradient with respect to $\mathbf{a}$ and equating it to $\mathbf{0}$ gives

$$\begin{aligned} \nabla L(\mathbf{a}, \lambda) &= \mathbf{0} & (2.66) \\ 2\boldsymbol{\Sigma}\mathbf{a} - 2\lambda\mathbf{a} &= \mathbf{0} & (2.67) \end{aligned}$$

where Eq. (2.67) can be rephrased into the eigenvalue-eigenvector problem:

$$\boldsymbol{\Sigma}\mathbf{a} = \lambda\mathbf{a} \tag{2.68}$$

Thus we see that the vector of scalar coefficients are in fact the eigenvectors of $\boldsymbol{\Sigma}$! Let $\boldsymbol{\Sigma}$ have the eigenvalue-eigenvector pairs

$$(\lambda_1, \mathbf{e}_1), (\lambda_2, \mathbf{e}_2), \ldots, (\lambda_l, \mathbf{e}_l) \quad \text{where} \quad \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_l \geq 0 \tag{2.69}$$

Since $\boldsymbol{\Sigma}$ is a covariance matrix (and thus is psd) it will have orthogonal eigenvectors such that

$$\mathbf{e}_i^T \mathbf{e}_j = 0 \qquad i \neq j \tag{2.70}$$

The second constraint, specified in Eq. (2.64), will also hold:

$$\text{Cov}\{y_i, y_j\} = \mathbf{e}_i^T \boldsymbol{\Sigma} \mathbf{e}_j = \mathbf{e}_i^T \lambda_j \mathbf{e}_j = \lambda_j \mathbf{e}_i^T \mathbf{e}_j = 0 \qquad i \neq j \tag{2.71}$$

Then the orthonormal eigenvectors of the covariance matrix are the vectors of scalar coefficients that gives us the principal components. Now we check which eigenvector corresponds to which principal component. Recall that the first principal component is the one that maximizes the variance without any constraint. From Eq. (2.59):

$$\text{Var}\{y_i\} = \mathbf{e}_i^T \boldsymbol{\Sigma} \mathbf{e}_i = \mathbf{e}_i^T \lambda_i \mathbf{e}_i = \lambda_i \mathbf{e}_i^T \mathbf{e}_i = \lambda_i \qquad i = 1, \ldots, l \tag{2.72}$$

Hence the first principal component corresponds to the one found from projecting $\mathbf{x}$ onto the eigenvector of $\mathbf{\Sigma}$ corresponding to the largest eigenvalue. Because of the definition in Eq. (2.69) we see that this is $\mathbf{e}_1$. The second principal component will correspond to $\mathbf{e}_2$ and so on, thus $\mathbf{a}_i = \mathbf{e}_i$, $i = 1, \ldots, l$.

Usually one chooses to project the data onto the s first principal components, where $s \leq l$ and $\lambda_s > 0$. One can then form a $s \times 1$ vector, $\mathbf{y}$, of these components:

$$\mathbf{y} = [y_1, y_2, \ldots, y_s]^T = \left[\mathbf{e}_1^T\mathbf{x}, \mathbf{e}_2^T\mathbf{x}, \ldots, \mathbf{e}_s^T\mathbf{x}\right]^T \tag{2.73}$$

So far we have assumed that the covariance matrix $\mathbf{\Sigma}$ is known. In practice this is seldom the case, and the covariance matrix must be estimated from the data. Given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, one can use for instance the unbiased estimator:

$$\hat{\mathbf{\Sigma}} = \mathbf{S}_{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \tag{2.74}$$

where $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$ is the sample mean vector. Then PCA is done on $\hat{\mathbf{\Sigma}}$ in exactly the same way. Once the covariance matrix has been estimated, the eigenvectors and eigenvalues are calculated. Then the data vectors are projected onto s first eigenvectors, where s is the desired number of dimensions. In addition to the dimensionality reduction, performing a PCA might lead to new interpretation of the data [20].

If the data is centered, that the sample mean is the at the origin $\bar{\mathbf{x}} = \mathbf{0}$, Equation (2.74) can be simplified to

$$\hat{\mathbf{\Sigma}} = \frac{1}{n-1}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^T \tag{2.75}$$

Centering can be done by subtracting the sample mean from each data vector.

2.5.2 The Kernel Trick

Despite its good properties, PCA is a linear algorithm and hence is not suited for nonlinear problems. Given a dataset, usually high-dimensional, $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \subset \mathbb{R}^l$, the goal is to compute n corresponding patterns, $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n\} \subset \mathbb{R}^s$ that provide an informative representation of the input data. The following two sections are based on [39, Section 6.7.1]. Given the dataset we make an implicit mapping into a Reproducing Kernel Hilbert Space (RKHS), H

$$\mathbf{x} \in \mathbb{X} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in H \tag{2.76}$$

A Hilbert space is an abstract vector space equipped with an inner product. It is *complete*, which means that the techniques of calculus can be used. In theory, this Hilbert space can be infinite dimensional, but we select $s \leq l$ principal components from each of the n data vectors, just like for regular PCA.

While having to perform a mapping to a potentially infinite dimensional space sounds like an impossible task, it turns out that we do not have to find the mapping function.

Theorem 1. *Mercer's Theorem. Let* $\mathbf{x} \in \mathbb{R}^l$ *and a mapping* $\boldsymbol{\phi}$

$$\mathbf{x} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in H$$

where H *is a Hilbert space. Let the inner product operation have an equivalent representation*

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle = K(\mathbf{x}, \mathbf{z}) \tag{2.77}$$

where $\langle \cdot, \cdot \rangle$ *denotes the inner product operation in* H*. Then* $K(\mathbf{x}, \mathbf{z})$ *is a symmetric continuous function satisfying the following condition:*

$$\int_{\mathbf{C}} \int_{\mathbf{C}} K(\mathbf{x}, \mathbf{z}) g(\mathbf{x}) g(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0 \tag{2.78}$$

for any $g(\mathbf{x})$, $\mathbf{x} \in \mathbf{C}$ *such that*

$$\int_{\mathbf{C}} g(\mathbf{x})^2 d\mathbf{x} < +\infty \tag{2.79}$$

where $\mathbf{C}$ *is a compact (finite) subset of* $\mathbb{R}^l$*. The opposite is always true; that is, for* any *any symmetric, continuous function* $K(\mathbf{x}, \mathbf{z})$ *satisfying Eq.* (2.77) *and Eq.* (2.78) *there exists a space in which* $K(\mathbf{x}, \mathbf{z})$ *defines an inner product! Such functions are also known as kernels and the space* H *as* Reproducing Kernel Hilbert Space (RKHS).

2.5.3 Kernel PCA

Just like for regular PCA, we want to find s principal components in H which preserves most of the variance in this space. To do this we must look at the covariance matrix in this space.

To simplify the calculations for the time being, we assume that the transformed data is centered ($\sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i) = \mathbf{0}$) in the kernel space. This limitation makes the covariance matrix in the kernel space is easier to estimate, similar to Eq. (2.75) for regular PCA. With centered data in H, the covariance

matrix in the kernel space can be written as

$$\mathbf{R} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_i)^T \tag{2.80}$$

Then the principal components can be found by solving the eigenvalue problem:

$$\mathbf{R}\mathbf{v} = \lambda\mathbf{v} \tag{2.81}$$

The eigenvectors, $\mathbf{v}$, must be in the span of the mapped data. So $\mathbf{v} \in \text{span}\{\boldsymbol{\phi}(\mathbf{x}_1), \phi(\mathbf{x}_2), ..., \phi(\mathbf{x}_n)\}$. This means that the eigenvectors can be expressed as a linear combination of the mapped data and for $\lambda \neq 0$:

$$\mathbf{v} = \sum_{j=1}^{n} a_j\boldsymbol{\phi}(\mathbf{x}_j) \tag{2.82}$$

By inserting the expressions from Eq. (2.80) and Eq. (2.82) into Eq. (2.81) we get

$$\lambda\sum_{j=1}^{n} a_j\boldsymbol{\phi}(\mathbf{x}_j) = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_i)^T\sum_{J=1}^{n} a_j\boldsymbol{\phi}(\mathbf{x}_j) \tag{2.83}$$

Now, we multiply with $\boldsymbol{\phi}(\mathbf{x}_k)^T$ from the left and get

$$\lambda\boldsymbol{\phi}(\mathbf{x}_k)^T\sum_{j=1}^{n} a_j\boldsymbol{\phi}(\mathbf{x}_j) = \frac{1}{n}\boldsymbol{\phi}(\mathbf{x}_k)^T\sum_{i=1}^{n} \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}_i)^T\sum_{j=1}^{n} a_j\boldsymbol{\phi}(\mathbf{x}_j) \tag{2.84}$$

This should hold for every $\boldsymbol{\phi}(\mathbf{x}_k)$, $k = 1, \ldots, n$ and hence we can form a system of equations based on Eq. (2.84). If we now define a $n \times n$ matrix $\mathbf{K}$ with elements

$$\mathbf{K}(i,j) \equiv \boldsymbol{\phi}(\mathbf{x}_i)^T\boldsymbol{\phi}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j) \qquad , i,j = 1, 2 \ldots, n \tag{2.85}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is a kernel function as defined in Mercer's theorem. Then we can formulate the system of equations based on Eq. (2.84) as

$$\lambda\mathbf{K}\mathbf{a} = \frac{1}{n}\mathbf{K}^2\mathbf{a} \tag{2.86}$$

where $\mathbf{a} \equiv [a_1, a_2, \ldots, a_l]$ is the vector of linear coefficients used to express $\mathbf{v}$ in terms of the mappings. $\mathbf{K}$ is called the *kernel matrix* or the *Gram matrix*. Since it is a covariance matrix (estimate), we know that it will be symmetric and positive semi-definite, and therefore invertible. Hence Eq. (2.86) can be simplified by multiplying with $\mathbf{K}^{-1}$ from the left [26]:

$$\mathbf{K}\mathbf{a} = \lambda n\mathbf{a} \tag{2.87}$$

Which is an eigenvector-eigenvalue problem. Let $\mathbf{K}$ have the eigenvalue-eigenvector pairs

$$(n\lambda_1, \mathbf{a}_1), (n\lambda_2, \mathbf{a}_2), \ldots, (n\lambda_l, \mathbf{a}_l) \quad \text{where} \quad \lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_l \geq 0 \tag{2.88}$$

Then, as for regular PCA, we select the eigenvectors corresponding to the s dominant eigenvalues where $s \leq n$ and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_s > 0$.

As with regular PCA we need to impose a normalizing constraint on the "original" eigenvectors, $\mathbf{v}_k$, $k = 1, \ldots, s$.

$$\begin{aligned} 1 &= \langle \mathbf{v}_k, \mathbf{v}_k \rangle = \left\langle \sum_{i=1}^{n} a_{ki} \boldsymbol{\phi}(\mathbf{x}_i) \, , \, \sum_{i=j}^{n} a_{kj} \boldsymbol{\phi}(\mathbf{x}_j) \right\rangle \\ &= \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ki} a_{kj} K_{i,j} \\ &= \mathbf{a}_k^T \mathbf{K} \mathbf{a}_k = n\lambda_k \mathbf{a}_k^T \mathbf{a}_k \qquad , k = 1, 2, \ldots, s \end{aligned}$$

Which gives that

$$\mathbf{a}_k^T \mathbf{a}_k = \frac{1}{n\lambda_k} \qquad , k = 1, 2, \ldots, s \tag{2.89}$$

So far we have assumed that the data have been centered in the RKHS. However, this is not generally the case and for the intuition of preserving the covariance in H we need center. A method for centering was presented in [34] and [26]. The centered kernel matrix $\hat{\mathbf{K}}$ can be expressed in terms of the original kernel matrix as

$$\hat{\mathbf{K}} = \mathbf{K} - \frac{1}{n} \mathbf{1}_{n \times n} \mathbf{K} - \frac{1}{n} \mathbf{K} \, \mathbf{1}_{n \times n} + \frac{1}{n^2} \mathbf{1}_{n \times n} \mathbf{K} \, \mathbf{1}_{n \times n} \tag{2.90}$$

where $\mathbf{1}_{n \times n}$ is a $n \times n$ matrix with all elements equal to 1. While the centering is necessary for the interpretation of preserving the covariance in H, KPCA can be performed without centering with a different result. Most expositions on KPCA uses centering [18], and unless otherwise noted, centering is done when KPCA is mentioned in this thesis.

The method can then be summarized as

1. Compute the kernel matrix with elements defined in Eq. (2.85) ; $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$, $i, j = 1, 2 \ldots, n$. Denote the resulting matrix as $\mathbf{K}_{\text{org}}$.

2. Optional: Center the kernel matrix as defined in Eq. (2.90) ; $\mathbf{K} = \mathbf{K}_{\text{org}} - \frac{1}{n} \mathbf{1}_{n \times n} \mathbf{K}_{\text{org}} - \frac{1}{n} \mathbf{K}_{\text{org}} \, \mathbf{1}_{n \times n} + \frac{1}{n^2} \mathbf{1}_{n \times n} \mathbf{K}_{\text{org}} \, \mathbf{1}_{n \times n}$.

3. Find the s dominant eigenvalues/eigenvectors of the kernel matrix $\mathbf{K}$; $(\lambda_k, \mathbf{a}_k)$, $k = 1, 2 \ldots, s$.

4. Perform the normalization defined in Eq. (2.89) ; $\mathbf{a}_k^T \mathbf{a}_k = \frac{1}{n\lambda_k}$, $k = 1, 2, \ldots, s$.

5. For each point of interest, $\mathbf{x}$, compute the projections onto the s dominant eigenvectors to generate the new $s \times 1$ representation $\mathbf{y} = [y_1, y_2, \ldots, y_s]^T$. These are the s first principal components in the kernel space, where $y_k \equiv \langle \mathbf{v}_k, \boldsymbol{\phi}(\mathbf{x}) \rangle = \sum_{i=1}^{n} a_{ki} K(\mathbf{x}_i, \mathbf{x})$ for $k = 1, 2, \ldots, n$.

In clustering applications the final step is done for every data vector $\mathbf{x} = \mathbf{x}_i$ to generate the new representations $\mathbf{y}_i$ for $i = 1, \ldots, n$. Because of this it is possible to rewrite this in terms of matrices. From linear algebra, we know that since $\mathbf{K}$ is symmetric, it may be eigendecomposed as $\mathbf{K} = \mathbf{E}\boldsymbol{\Lambda}\mathbf{E}^T$, where $\boldsymbol{\Lambda}$ is a diagonal matrix containing all eigenvalues $\lambda_1, \ldots, \lambda_n$ of $\mathbf{K}$ and $\mathbf{E}$ is a matrix with the corresponding eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ as columns [2, Chap. 7.3].

If we organize the transformed dataset $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ into a $n \times s$ matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{bmatrix}$$

where $\mathbf{y}_j$, $j = 1, \ldots, n$ are $s \times 1$ vectors. A projection onto s KPCA axes given by the eigenvectors corresponding to the largest eigenvalues can be expressed as

$$\mathbf{Y} = \boldsymbol{\Lambda}_s^{\frac{1}{2}} \mathbf{E}_s^T \tag{2.91}$$

where $\boldsymbol{\Lambda}_s$ is a $s \times s$ diagonal matrix with the largest eigenvalues of $\mathbf{K}$ and $\mathbf{E}_s$ is a $n \times s$ matrix with the corresponding eigenvectors as columns.

Then a clustering *algorithm* is applied to the transformed dataset $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ given as the rows of $\mathbf{Y}$.

Figure 2.12 shows the original data in the feature space to the left while the right plot is of the first data projected on the two first principal axes found by kernel PCA. We notice that the original dataset would have presented a very challenging clustering problem. After the data transformation we would expect that relatively simple clustering algorithms, like k-means, could separate the two clusters.

The kernel used for all these examples is the Gaussian kernel with a single bandwidth parameter (covariance matrix proportional to the identity

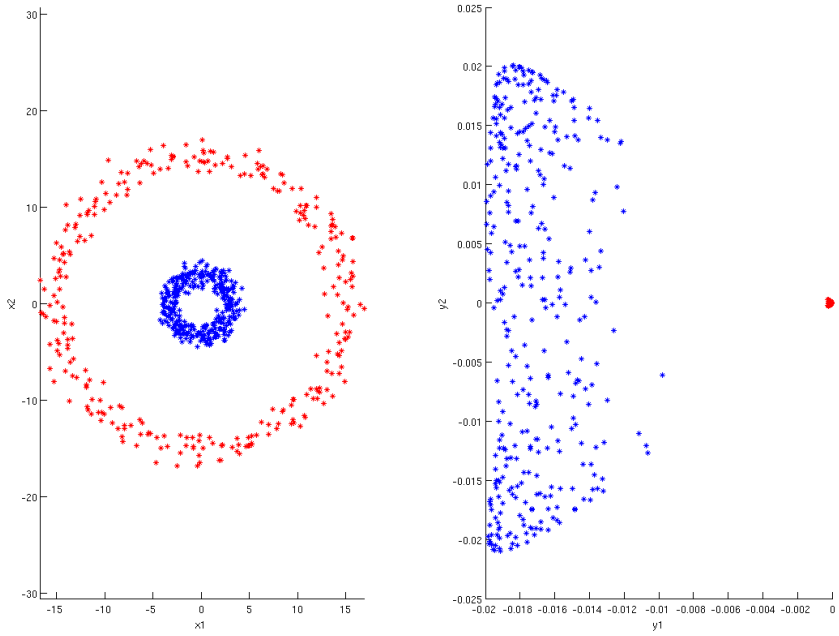


Figure 2.12: Two concentric circles with noise.

matrix), used for density estimation in Eq. (2.53) in Section 2.4.1:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{(2\pi)^{l/2} h^l} \exp\left(-\frac{1}{2}\left|\left|\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right|\right|^2\right) \tag{2.92}$$

where $||\cdot||$ is the Euclidean norm, l is the number of variables in each feature vector $\mathbf{x}$, and h is the bandwidth. The expression outside the exponential is constant for a dataset given a bandwidth and does not affect the result. Hence the Gaussian kernel can be written as

$$K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\left|\left|\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right|\right|^2\right) \tag{2.93}$$

The result of kernel PCA on the right in Fig. 2.13 may seem like it still might be too much of a challenge for k-means. However, we should recall that this is only the projection onto the two first components.

Figure 2.14 shows the projection onto the first three components. From this perspective the clusters seem easier to separate. Note that using only the first two components might be sufficient for slightly more advanced clustering

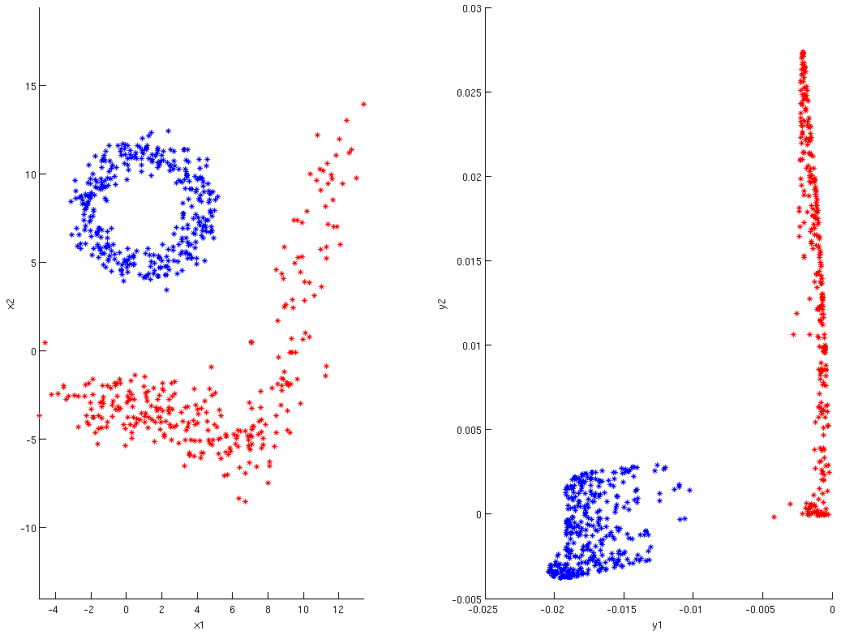


Figure 2.13: Circle and "reverse L" shape.

methods, for instance mean shift, to provide a good result. It should also be mentioned that for structures like the ones seen in Fig. 2.13 is where hierarchical clustering methods shine.

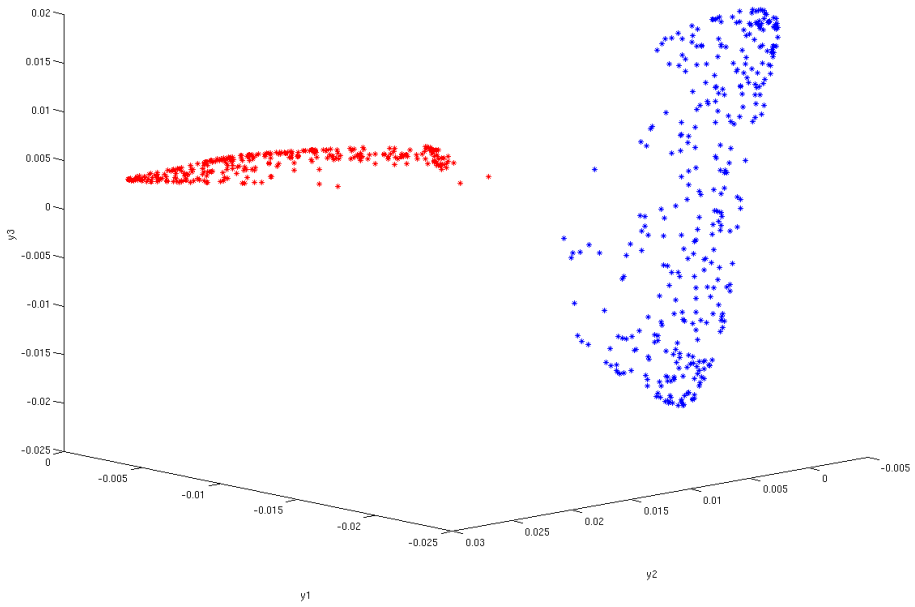


Figure 2.14: Projection onto first 3 components.

2.6 Laplacian Eigenmaps

Laplacian eigenmaps is a graph based method for dimensionality reduction which tries to preserve neighbourhood information. This will implicitly emphasize the natural clusters in the dataset [4]. The data points are thought of as vertices in a graph, with edges between neighbouring points.

Given a proximity (similarity or dissimilarity) measure, the proximity between each data point is calculated. Based on these proximity measures one finds points that are close and label them as neighbours. An edge is defined between every pair of neighbours. There are two main approaches to defining which points are close. One defines all points within a certain (user defined) distance of each other as close, while the other says that each point has a fixed (user defined) number of neighbours. Each approach has its own merits.

Then a weight is defined for every edge to create the weighted *adjacency matrix*. Typically, the choice of weight function depends on how closeness was defined earlier. Often a a Gaussian function is used for this, assigning large weights to neighbours with a small Euclidean distance between them.

Based on the weighted adjacency matrix, a graph Laplacian matrix is

defined. In [41] it was pointed out that there is some inconsistency in the scientific community regarding which matrix is called the "graph Laplacian", and the article discussed the three different conventions. While the three definitions give different result, they are related.

Once a graph Laplacian as been chosen, the eigenvalues and eigenvectors are calculated. Then the new representation of the data is found as components of the eigenvectors corresponding to the smallest eigenvalues. The number of eigenvectors used is equal to the desired dimension of the transformed data. Since the clustering procedure is based on solving an eigenvalue problem it is a spectral clustering algorithm.

This section is based on [41].

2.6.1 Defining The Graph

Vertices and Edges

Given a set of data vectors $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ of dimension $l \times 1$, an adjacency graph is constructed. The graph consists of vertices (also called nodes), V, and edges, E. The set of vertices is $V = \{v_i\,, i = 1, 2, ..., n\}$ where each v_i equals the corresponding data vector $\mathbf{x}_i$. $E = \{e_{ij}\}$ is a set of edges connecting v_i and v_j. The edges only connect points that are close in some sense. The graph is then denoted $G(V, E)$.

There are two main approaches to defining which nodes are close to each other, and hence where to put edges in the adjacency graph. Both methods have its advantages and disadvantages.

ϵ-neighbourhoods: Vertices v_i and v_j are connected if the distance between them is less than a user defined parameter $\epsilon \in \mathbb{R}$. A often used criterium is $||\mathbf{x}_i - \mathbf{x}_j||^2 < \epsilon$ where $||\cdot||^2$ is the squared Euclidean norm in $\mathbb{R}^l$.

k nearest neighbour: Vertex v_i is connected to vertex v_j if $\mathbf{x}_j$ is one of the k nearest neighbours (kNN) of $\mathbf{x}_i$ where $k \in \mathbb{N}$ is a user defined parameter. However, this definition leads to neighbourhood relationships that are not symmetric. This means that while vertex v_j is one of the k nearest neighbours of vertex v_i, v_i might not be one of the k nearest neighbours of vertex v_j. The graph is said to be *directed*. For Laplacian eigenmaps the graph should be *undirected*, which will make the adjacency matrix symmetric. There are two ways of achieving this. The first is that v_i and v_j are connected if v_i is one of the kNN of v_j OR v_j is one of the kNN of v_i. This is usually just called the *k-nearest neighbour graph* [41]. The second approach is called the *mutual k-nearest*

neighbour graph where v_i and v_j are connected if v_i is one of the kNN of v_j AND v_j is one of the kNN of v_i.

The first method is geometrically motivated and the adjacency graph will be naturally symmetric. This approach may often, depending on the parameter ϵ, lead to a graph consisting of one or more subgraphs with no connection to each other. A graph (or subgraph) is said to be a *connected component* if you by starting at any one of the vertices can reach any other vertex by "moving along the edges between vertices". The ϵ parameter can be difficult to chose, especially if nothing is known about the data in advance.

The number of nearest neighbours is easier to choose and it tends to lead to connected graphs. This approach is less intuitive as actual distance between vertices which are considered close can vary greatly depending on how dense the local concentration of data points is. However, this can be an advantage if the dataset contains regions of different densities. With the *k-nearest neighbour graph* (rather than the *mutual k-nearest neighbour graph*), each point will always be connected with at least $k-1$ other points (not k since a point is always considered to be its own neighbour).

To illustrate this, consider this toy data set. The data points are marked

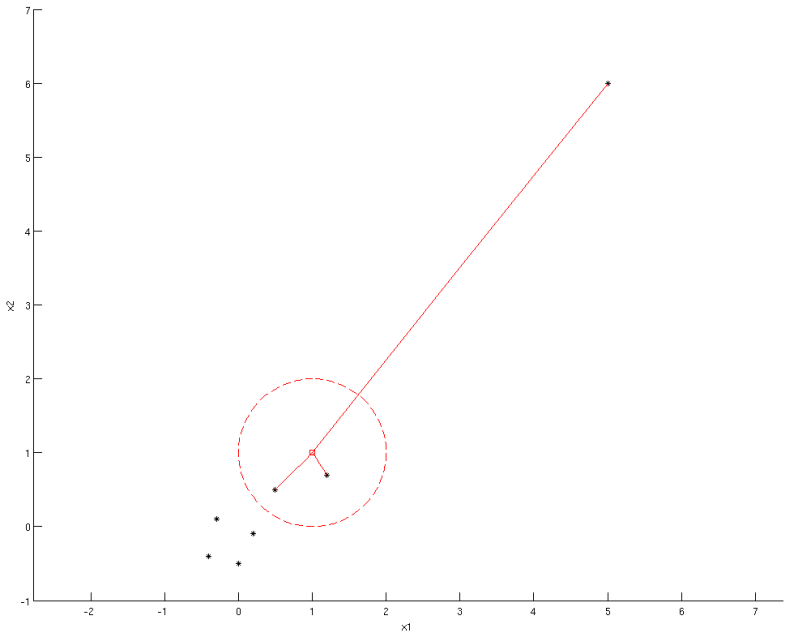


Figure 2.15: Neighbouring points example.

by black dots, except for our point of interest which is marked by a red square. The dashed red line is the circle centered at the point of interest with radius $\epsilon = 1.5$. The two point inside the radius will then be considered neighbours. The outlier in the top right corner is not connected with the rest of the graph

The solid red lines shows neighbours if we construct the graph with the kNN approach with $k = 3$. An edge is put between our point of interest and an outlier, even though there are points that are much closer that do not get an edge.

The Weight of an Edge and the Degree of a Vertex

Now we assign a weight $w_{ij} > 0$ to each edge e_{ij} to construct the $n \times n$ weighted adjacency matrix $\mathbf{W}$. The only non-zero elements of this matrix are those corresponding to edges. So

$$w_{ij} = \begin{cases} W(\mathbf{x}_i, \mathbf{x}_j), \text{ if there is an edge}, e_{ij}\,, \text{ between } v_i \text{ and } v_j \\ 0, \text{ otherwise} \end{cases} \tag{2.94}$$

where $w_{ij}\,,\ i, j = 1, \ldots, n$ are the elements of $\mathbf{W}$ and $W(\mathbf{x}_i, \mathbf{x}_j)$ is a weight function. A weight function must be symmetric so $W(\mathbf{x}_i, \mathbf{x}_j) = W(\mathbf{x}_j, \mathbf{x}_i)$. Since the graph is undirected, the weight matrix will be symmetric, $w_{ij} = w_{ji} \Rightarrow \mathbf{W} = \mathbf{W}^T$.

The choice of neighbourhood definition in Section 2.6.1 should influence the choice of weight function. When using k nearest neighbours, the distance between connected vertices can vary greatly. The weight function should then assign a high weight to edges between points that are also close geometrically. A typical choice is to use a Gaussian, called a heat kernel in [4], to weight the edges. Then $W(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}}$ where $\sigma^2 \in \mathbb{R}_+$ is a user defined parameter.

When ϵ-neighbourhoods are used, the distance between neighbours is less than ϵ for the chosen distance measure and a weight function is less important. Therefore it might not be necessary to use a complicated weight function and edges might be weighted equally: $W(\mathbf{x}_i, \mathbf{x}_j) = 1\,,\ \forall\, \mathbf{x}_i, \mathbf{x}_j$. Notice that this is equivalent to the heat kernel when $\sigma^2 \to \infty$.

We now define the degree of a vertex $v_i \in V$ as the sum of all its associated weights

$$d_i = \sum_{j=1}^{n} w_{ij} \tag{2.95}$$

The only non-zero contributions to sum in Eq. (2.95) will be from neighbours

of vertex v_i. The *degree matrix* is then defined as a $n \times n$ diagonal matrix $\mathbf{D}$ with elements $d_1, d_2, \ldots, d_n$ on the diagonal.

2.6.2 Graph Laplacian

The motive for defining the graph is that we wish to perform a mapping to a lower dimensional space, that preserves the structure of the graph $G(V, E)$ defined by the weighted adjacency matrix $\mathbf{W}$. Given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, an embedding in m-dimensional Euclidean space is then given by

$$\mathbf{x}_i \in \mathbb{R}^l \mapsto \mathbf{y}_i \in \mathbb{R}^m \, , \; i = 1, 2, ..., n \tag{2.96}$$

There are several interpretations of the algorithm that follows. Laplacian eigenmaps can be considered from graph partitioning, random walk and perturbation perspectives [41].

In [4] the problem was formulated as trying to to construct *"... a representation for data lying on a low-dimensional manifold embedded in a high-dimensional space."*. The algorithm was then justified by considering the weighted Laplacian of the adjacency graph as an approximation for the Laplace Beltrami operator, which provides an optimal embedding for the manifold.

A simpler interpretation is that we wish to preserve local neighbourhood information in a way that is optimal in some sense. In other words, if nodes are part of the same neighbourhood in the original space, they should be mapped close to each other. We will now see how the algorithm can be derived from optimizing a cost function.

Cost Function Optimization

Now, let $\mathbb{X}$, defined above, be organized in a $n \times l$ matrix $\mathbf{X}$ where rows correspond to data vectors (observations) and the columns to the data dimensions (variables). With the mapping be defined in Eq. (2.96), the mapped data can be organized in a $n \times m$ matrix $\mathbf{Y}$ with rows $\mathbf{y}_i^T = [y_{i1}, y_{i2}, \ldots, y_{im}]$.

We first consider the s'th of the m components of $\mathbf{y}_i$, y_{is} where $i = 1, \ldots, n$. We define a cost function as

$$E = \sum_{s=1}^{m} E_s \tag{2.97}$$

where

$$E_s = \sum_{i=1}^{n} \sum_{j=1}^{n} (y_{is} - y_{js})^2 w_{ij} \tag{2.98}$$

is the cost associated with component (column) s of $\mathbf{Y}$.

To minimize Eq. (2.97) we have to minimize Eq. (2.98) for $s = 1, \ldots, m$. We notice that when the data vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ are close in the l-dimensional space, the edge weight w_{ij} will be positive and to minimize the cost function in Eq. (2.98) we have to minimize $(y_{is} - y_{js})^2$ for all $s = 1, \ldots, m$, which corresponds to mapping the nodes close to each other. Points that are not neighbours will not affect the cost since the respective weights are zero. With some algebra, Eq. (2.98) can be expressed it on another form:

$$\begin{aligned} E_s &= \sum_{i=1}^{n}\sum_{j=1}^{n}(y_{is} - y_{js})^2 w_{ij} \\ &= \sum_{i=1}^{n}\left[y_{is}^2 \sum_{j=1}^{n} w_{ij}\right] + \sum_{j=1}^{n}\left[y_{js}^2 \sum_{i=1}^{n} w_{ij}\right] - 2\sum_{i=1}^{n}\sum_{j=1}^{n} y_{is}y_{js}w_{ij} \\ &= 2\sum_{i=1}^{n}\left[y_{is}^2 \sum_{j=1}^{n} w_{ij}\right] - 2\sum_{i=1}^{n}\sum_{j=1}^{n} y_{is}y_{js}w_{ij} \\ &= 2\sum_{i=1}^{n} y_{is}^2 d_{ii} - 2\sum_{i=1}^{n}\sum_{j=1}^{n} y_{is}y_{js}w_{ij} \end{aligned}$$

which can be formulated in vector form by denoting the s'th column vector of $\mathbf{Y}$ as $\tilde{\mathbf{y}}_s = [y_{1s}, y_{2s}, ..., y_{ns}]^T$. The tilde is used to distinguish them from the row vectors of $\mathbf{Y}$ which correspond to the mapping of each of the n data points.

$$E_s = 2\left(\tilde{\mathbf{y}}_s^T \mathbf{D}\tilde{\mathbf{y}}_s - \tilde{\mathbf{y}}_s^T \mathbf{W}\tilde{\mathbf{y}}_s\right) \tag{2.99}$$

The *unnormalized graph Laplacian matrix*, $\mathbf{L}$, is then defined as

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{2.100}$$

and Eq. (2.99) can be written as

$$E_s = 2\tilde{\mathbf{y}}_s^T \mathbf{L}\tilde{\mathbf{y}}_s \tag{2.101}$$

The obvious minimum of E_s is the trivial solution $y_{is} = 0\,, \forall i, s$, which corresponds to mapping all points to 0. To avoid this problem, we constrain $\tilde{\mathbf{y}}_s$ to a specific norm. We will see that the choice of this constraint will determine the type of Laplacian matrix we are working on.

The Unnormalized Graph Laplacian

We now impose the constraint

$$\tilde{\mathbf{y}}_s^T \tilde{\mathbf{y}}_s = 1 \tag{2.102}$$

This choice actually implies that we are working on what is called the unnormalized graph Laplacian matrix. Lagrange optimization can be used to minimize each E_s subject to the constraint . The Lagrangian function, f, can then be written as

$$f(\tilde{\mathbf{y}}_s, \lambda_s) = \tilde{\mathbf{y}}_s^T \mathbf{L} \tilde{\mathbf{y}}_s - \lambda_s \left(\tilde{\mathbf{y}}_s^T \tilde{\mathbf{y}}_s - 1 \right) \tag{2.103}$$

evaluating all the partial derivatives of $\tilde{\mathbf{y}}_s$ and solving when equal to zero gives us

$$\mathbf{L} \tilde{\mathbf{y}}_s = \lambda_s \tilde{\mathbf{y}}_s \tag{2.104}$$

which we recognize as an eigenvalue-eigenvector problem. We can now insert this expression into Equation (2.101):

$$E_s = 2 \tilde{\mathbf{y}}_s^T \mathbf{L} \tilde{\mathbf{y}}_s = 2 \tilde{\mathbf{y}}_s^T \lambda_s \tilde{\mathbf{y}}_s = 2 \lambda_s \tag{2.105}$$

So E_s is minimized by selecting $\tilde{\mathbf{y}}_s$ as the eigenvector corresponding to the smallest eigenvalue of $\mathbf{L}$. However, we also want to impose an additional constraint on the component vectors of the mapped dataset $\tilde{\mathbf{y}}_s$, $s = 1, \ldots, m$.

That is that they should not be linearly dependent. If all $\tilde{\mathbf{y}}_s$, $s = 1, \ldots, m$ were set equal to the eigenvector corresponding to the smallest eigenvalue, the mapping would be to a one dimensional line in $\mathbb{R}^m$. In general, if any of the $\tilde{\mathbf{y}}_s$'s were linearly dependent, the mapping would span less than the m dimensions we specified.

It is clear that we need another constraint. As the matrix $\mathbf{L}$ is symmetric and positive semi-definite, it will have non-negative eigenvalues and orthogonal eigenvectors. Hence we impose the restriction that the component vectors should be orthogonal $\tilde{\mathbf{y}}_s$, $s = 1, \ldots, m$.

Then minimizing the overall cost function E as defined in Eq. (2.97) is done by defining $\mathbf{Y}$ as having column vectors:

$$\mathbf{Y} = \left[\begin{array}{c|c|c|c} \tilde{\mathbf{y}}_1 & \tilde{\mathbf{y}}_2 & \ldots & \tilde{\mathbf{y}}_m \end{array} \right] \tag{2.106}$$

where the column vectors $\tilde{\mathbf{y}}_s$, $s = 1, \ldots, m$ are selected as the solutions of the eigenvector-eigenvalue problem

$$\mathbf{L} \tilde{\mathbf{y}} = \lambda \tilde{\mathbf{y}} \tag{2.107}$$

corresponding to the m smallest *meaningful* eigenvalues of the unnormalized graph Laplacian matrix $\mathbf{L}$. The definition of meaningful will be explained in the following section.

Figure 2.16 shows a toy data example similar to the one in Fig. 2.12, the circles are defined with the same radius and Gaussian noise, but fewer

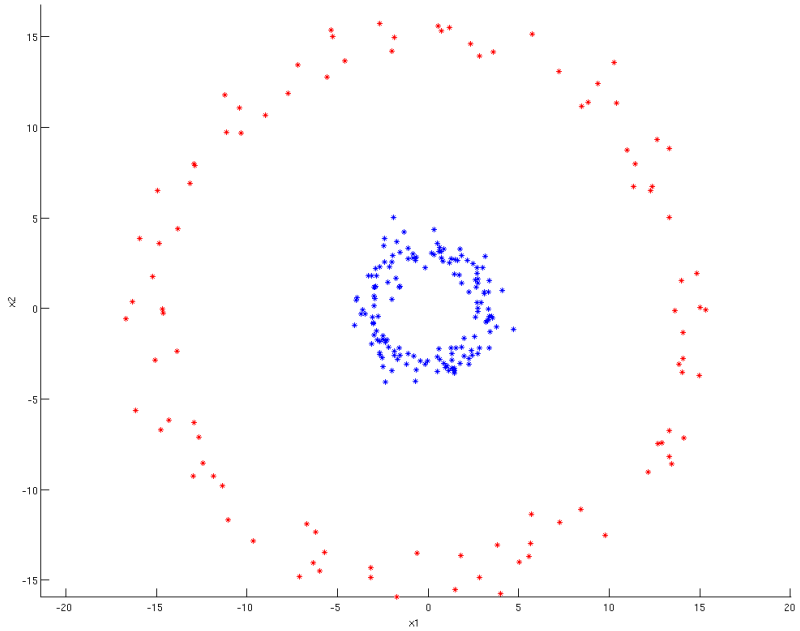


Figure 2.16: Ring data.

data points are used. This is to illustrate the difference between the way the neighbours are defined.

The unnormalized graph Laplacian was then constructed based on the squared Euclidean distance $||\cdot||^2$ used as the proximity measure on the data in Fig. 2.16. The result of Laplacian eigenmaps for $m = 2$ using both ϵ-neighbourhood and k nearest neighbours are shown in Fig. 2.17. The edges were weighted by a Gaussian weight function with $\sigma = 1$ which gives $W(\mathbf{x}_i, \mathbf{x}_j) = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}} = e^{-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2}}$.

The two subplots in the top row was made based on the ϵ-neighbourhood method with $\epsilon = 0.75$ to the left and $\epsilon = 2$ to the right. The smallest ϵ value will produce a graph with several connected components and the mapping is unable to separate the two rings. With $\epsilon = 2$ the blue ring is mapped to a single point, while the red ring seems to be mapped to three different points.

It would be possible to accurately cluster this mapping, depending on the clustering method used. An ad hoc method for this case could be to assign all points with a non-zero first component to one cluster and the rest to the second cluster. Increasing ϵ will not give a better separation in this particular case, and when it becomes large enough it will start connecting

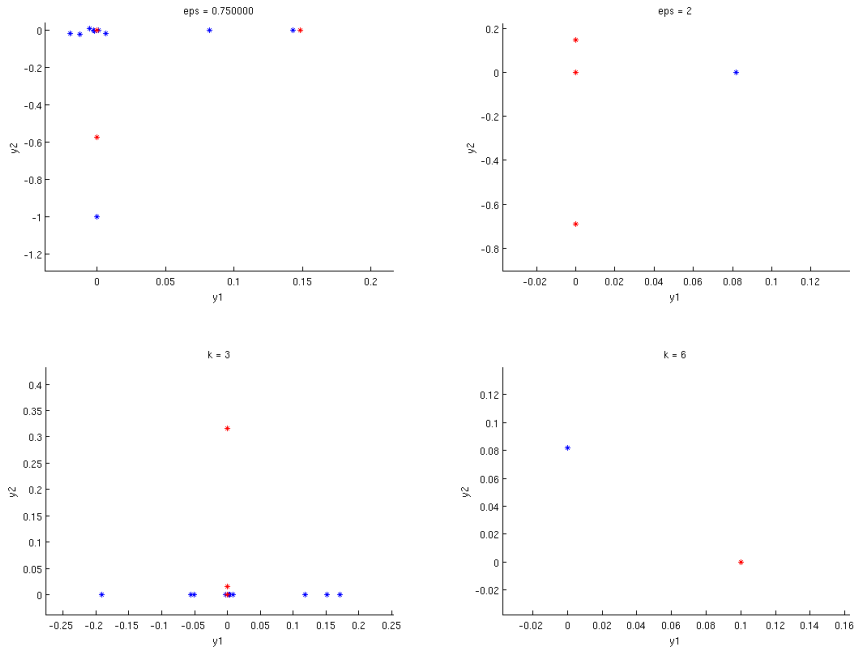


Figure 2.17: First two components of transformation.

the inner and outer circle.

The second row shows the result of using kNN, with $k = 3$ for the bottom left corner. Clearly this mapping is not good, and this is caused by the low number of neighbours for each point. In the bottom right corner k is increased to 6, and the result is two connected components mapped to two different points. This is ideal for clustering. Even though the outer circle is somewhat sparsely populated, the structure is uncovered by using only the 6 nearest neighbours. Increasing k to 10 gives the exact same result.

Figure 2.17 illustrates some of the advantages of kNN over ϵ-neighbourhoods; it handles regions of different densities better, it tends to lead to fewer connected components in the graph and the parameter k is easier to select.

Connected Components of a Graph

$\mathbf{L}$ will always have $\lambda = 0$ as its smallest eigenvalue, if the graph is fully connected this corresponds to the constant $n \times 1$ one vector, $\mathbf{1} = [1, 1, \ldots, 1]^T$. Here the normalization constraint has been left out. This is can easily be verified by looking at Eq. (2.98).

The eigenvector $\mathbf{1}$ (or any vector proportional to it) will not provide any

distinction between the points if it was used for mapping as it would imply that $\mathbf{y}_{11} = \mathbf{y}_{21} = \ldots = \mathbf{y}_{n1} = \alpha$ where $\alpha \in \mathbb{R}$ is a normalization constant. Hence the eigenvector $\tilde{\mathbf{y}} = \mathbf{1}$ corresponding to the eigenvalue $\lambda = 0$ is not considered meaningful in this case. However, the situation is different if the graph is not fully connected.

Theorem 2. *Number of connected components and the spectrum of* $\mathbf{L}$*. Let* G *be an undirected graph with non-negative weights. Then the multiplicity of the eigenvalue* 0 *of* $\mathbf{L}$ *equals the number of connected components* $A_1, \ldots, A_r$ *in the graph. The eigenspace of eigenvalue* 0 *is spanned by the indicator vectors* $\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_r}$ *of those components.*

The indicator vectors $\mathbf{1}_{A_i}$, $i = 1, \ldots, r$ are $n \times 1$ vectors with elements

$$\mathbf{1}_{A_i}(j) = \begin{cases} 1, \text{if } v_j \in A_i \\ 0, \text{otherwise} \end{cases} \tag{2.108}$$

Thus if a graph consists of several subgraphs (more than one connected component), as would be the case if we chose the ϵ-neighbourhood marked in Fig. 2.15, the eigenvectors corresponding to $\lambda = 0$ will indicate which points belong to different subgraphs. If the graph is well defined, this will provide a good clustering solution! This was seen in Fig. 2.17 where the eigenvectors for eigenvalue zero was used. However, if the we make a poor choice of neighbourhood parameter (k or ϵ) we will end up with many sparsely populated clusters.

A proof of Theorem 2 can be found in [41].

Normalized Graph Laplacian

As mentioned the constraint in Eq. (2.102) is associated with the unnormalized graph Laplacian. Another common constraint is

$$\tilde{\mathbf{y}}_s^T \mathbf{D} \tilde{\mathbf{y}}_s = 1 \tag{2.109}$$

which will lead to optimization of the following Lagrange function, with the only difference from Eq. (2.103) being that the degree matrix $\mathbf{D}$ being included in the constraint

$$f(\tilde{\mathbf{y}}_s, \lambda_s) = \tilde{\mathbf{y}}_s^T \mathbf{L} \tilde{\mathbf{y}}_s - \lambda_s \left(\tilde{\mathbf{y}}_s^T \mathbf{D} \tilde{\mathbf{y}}_s - 1 \right) \tag{2.110}$$

Minimizing the overall cost function, E, will be done by solving the *generalized* eigenvalue-eigenvector problem:

$$\mathbf{L}\tilde{\mathbf{y}} = \lambda \mathbf{D} \tilde{\mathbf{y}} \tag{2.111}$$

Again we solve for m eigenvectors corresponding to the smallest (meaningful) eigenvalues. Since the degree matrix $\mathbf{D}$ is diagonal with non-zero diagonal elements, the generalized eigenvalue-eigenvector problem is equivalent to solving:

$$\mathbf{D}^{-1}\mathbf{L}\tilde{\mathbf{y}} = \lambda\tilde{\mathbf{y}} \tag{2.112}$$

From this it is possible to define a normalized Laplacian matrix directly as $\mathbf{D}^{-1}\mathbf{L}$. The problem can also be reformulated to obtain a second definition of the normalized Laplacian [39]. So there are two matrices called the normalized graph Laplacian:

$$\mathbf{L}_{\text{rw}} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W} \tag{2.113}$$

$$\mathbf{L}_{\text{sym}} = \mathbf{D}^{-1/2}\,\mathbf{L}\,\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\,\mathbf{W}\,\mathbf{D}^{-1/2} \tag{2.114}$$

The first matrix, which is the one that follows directly from the constraint in Eq. (2.111), is denoted by $\mathbf{L}_{\text{rw}}$ as it is closely related to a random walk, the second by $\mathbf{L}_{\text{sym}}$ as it is a symmetric matrix [41].

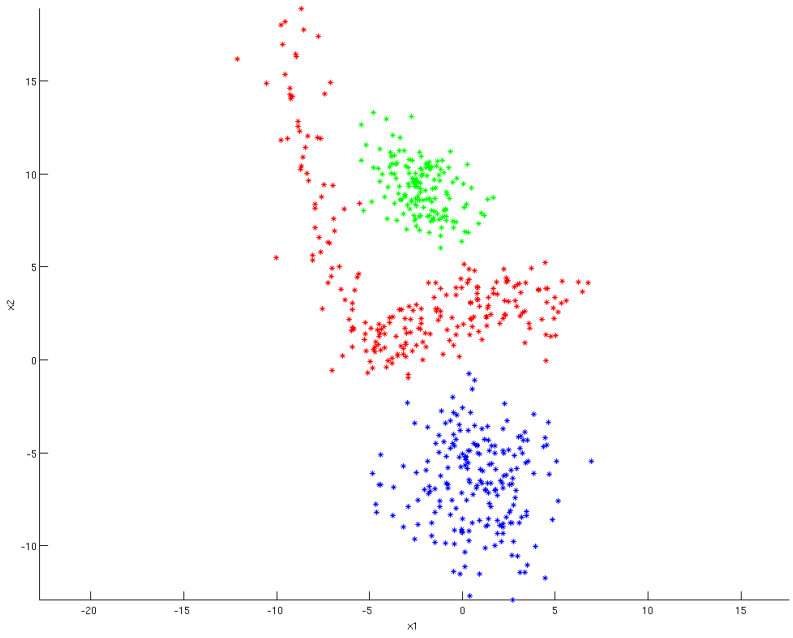


Figure 2.18: Three groups of data.

Figure 2.18 is another toy data example with three different groups. Clustering this dataset into the clusters corresponding to the three colours would

clearly be a very difficult task. However, this example is just meant to illustrate that there is a difference when using the normalized $\mathbf{L}_{\text{rw}}$ instead of the unnormalized graph Laplacian $\mathbf{L}$.

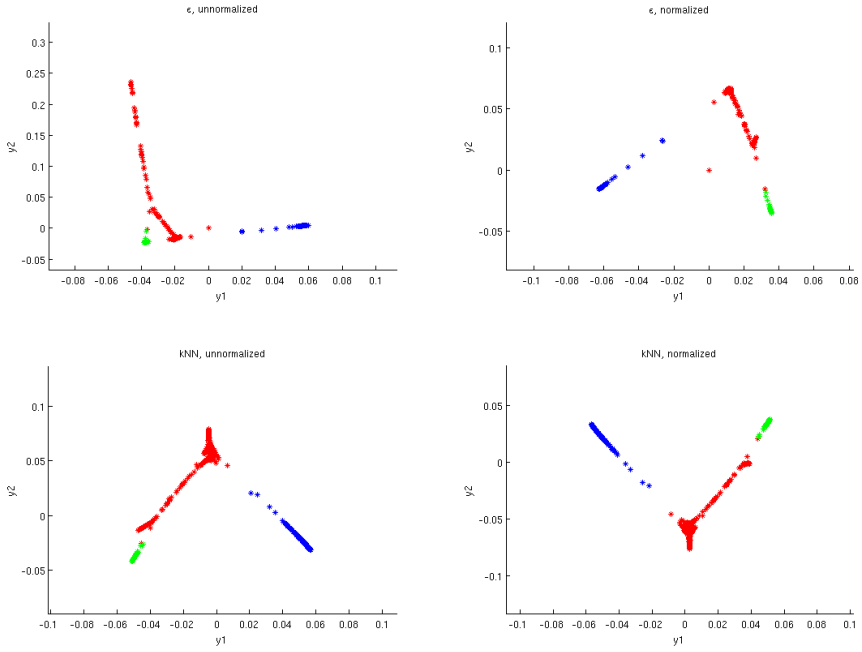


Figure 2.19: Four different eigenmaps.

The top row of Fig. 2.19 shows the first two components for the graph constructed using an ϵ-neighbourhood with $\epsilon = 4.0$. The bottom row is based on the graph constructed with k nearest neighbours for $k = 10$.

Like the example for Fig. 2.17, proximity was given by the squared Euclidean distance and the edges were weighted by a Gaussian weight function with $\sigma = 1$. The first two components of the mapping is plotted. As this graph consisted of a single connected component, the eigenvalue $\lambda = 0$ had multiplicity one and was ignored.

The two plots on the left were based on $\mathbf{L}$, while those on the right were based on $\mathbf{L}_{\text{rw}}$. We see that there is a difference for the ϵ-neighbourhood case, where the result based on the normalized matrix seems similar to the result from kNN. While the two plots based on kNN look quite different at first, it seems like the normalized plot might be obtained by rotation the plot based on the unnormalized graph Laplacian by 180°.

Figure 2.20 is the same as the bottom row of Fig. 2.19, only here $k = 25$. This figure illustrates that the normalization is dependent on the parameters.

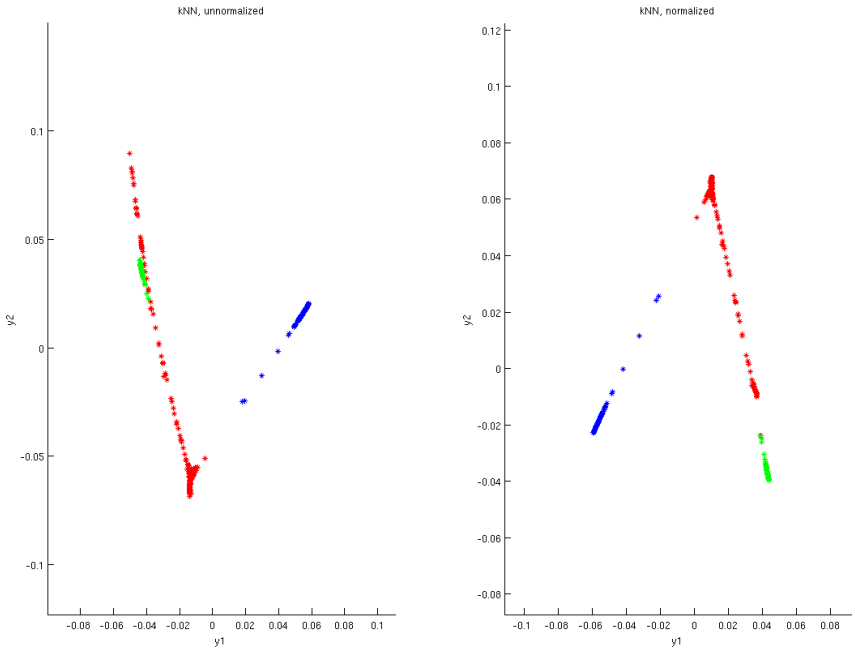


Figure 2.20: kNN with $k = 25$.

While normalization for this simple example seems to provide better results, this might not always be the case.

Which one of the three matrices $\mathbf{L}$, $\mathbf{L}_{\text{rw}}$ or $\mathbf{L}_{\text{sym}}$ to use will not be discussed here. While $\mathbf{L}_{\text{rw}}$ and $\mathbf{L}_{\text{sym}}$ have the same eigenvalues, their eigenvectors are different. It should be mentioned that [41] argues for using a normalized graph Laplacian and using $\mathbf{L}_{\text{rw}}$ rather than $\mathbf{L}_{\text{sym}}$.

2.6.3 Laplacian Clustering Methods

So far we have assumed that the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ is available, but this is not strictly necessary. All that is needed to run the algorithm is a matrix of proximity (similarity or dissimilarity) measures. As long as we assume that the weights we choose can be calculated based on the proximity measure we are given, the dataset is not actually needed.

Hence we assume that the input to the algorithm is a $n \times n$ matrix of proximities, $\mathbf{P}$. If no $\mathbf{P}$ is given it is calculated from the dataset $\mathbb{X}$. Let $\hat{\mathbf{L}}$ denote either the unnormalized $\mathbf{L}$ or the normalized $\mathbf{L}_{\text{rw}}$ graph Laplacian matrix. The algorithm for a clustering method based on Laplacian eigenmaps can then be summarized as

1. Find the neighbours from $\mathbf{P}$ using either the ϵ-neighbourhood or k nearest neighbours approach as described in Section 2.6.1.

2. Weight each edge found in Step 1 by a weight function to create the weighted adjacency matrix $\mathbf{W}$ and sum the weights of each vertex to find the degree matrix $\mathbf{D}$ as defined in Section 2.6.1.

3. Compute the unnormalized Laplacian matrix $\mathbf{L}$ by using Eq. (2.100). If normalization is desired, use Eq. (2.113). Either way, denote the result $\hat{\mathbf{L}}$.

4. Compute the first m eigenvectors, $\tilde{\mathbf{y}}_i$, $i = 1, \ldots, m$, of $\hat{\mathbf{L}}$ corresponding to the smallest eigenvalues, ignoring the eigenvalue $\lambda = 0$ if it has multiplicity one.

5. Construct the $n \times m$ matrix $\mathbf{Y}$ with columns $\tilde{\mathbf{y}}_i$, $i = 1, \ldots, m$. A mapping $\mathbf{x}_j \in \mathbb{R}^l \mapsto \mathbf{y}_j \in \mathbb{R}^m$ is given by the rows of $\mathbf{Y}$: $\mathbf{y}_j$, $j = 1, \ldots, n$.

6. Cluster the points given as the rows, $\mathbf{y}_j$, $j = 1, \ldots, n$, of $\mathbf{Y}$ by some simple clustering algorithm, for instance by using k-means. If $\lambda = 0$ has multiplicity greater than one, a clustering solution with the number of clusters equal to the multiplicty of the zero eigenvalue is given directly by looking at the columns of $\mathbf{Y}$ as cluster indicator vectors.

An algorithm based on $\mathbf{L}_{\text{sym}}$ requires an additional normalization of the rows of $\mathbf{Y}$ to norm 1. The other steps of the algorithm are the same. Another important aspect of Laplacian eigenmaps is that while the matrix in the eigenvalue-eigenvector problem can be quite big ($n \times n$), it is usually sparse as each point should have few neighbours relative to the number of overall data points n.

Chapter 3

Information Theoretical Learning

3.1 Introduction

Information Theoretical Learning (ITL) is a generic term used to describe the use of information theory descriptors of entropy and divergence in machine learning, signal processing and data analysis [30]. The ITL concept focuses on a particular type of Renyi entropy which can be easily estimated by kernel density estimation (KDE). This leads to several clustering methods and algorithms with relatively simple entropy-based cost and dissimilarity functions [30].

The theory of *information* was developed by Shannon in [35] and has found applications in many areas. Information theory provides scalar descriptors of probability density functions (pdfs) and the statistical similarity between them. In the terminology used in [30] this is the *mesoscopic modeling level*. Statistical moments[1] are the most used mesoscopic descriptors of the pdf and often *learning* from the dataset is based on these.

Traditional methods in these fields often emphasizes second-order moments when selecting an optimality criterion for learning [8]. This is closely related with the Gaussian distribution, which is completely characterized by its first and second order moments. There are several reasons for this emphasis. One is that cost functions based on second-order statistics often leads to simple and intuitive learning algorithms with relatively low computational complexity. Another important reason is that the Gaussian distribution has been a good model for many real world problems.

However, with the ever increasing number of problems and fields where information can be quantized, it is clear that Gaussianity of the data should

[1]The most widely used moments are the expectation and the variance, which are the first and second order moments. See [42, Chapter 7] for a brief description of moments.

be considered a special case and not the default assumption. In clustering, the emphasis on second-order statistics is most apparent in non-spectral methods, but also in Kernel Principal Component Analysis (KPCA) the method seeks to preserve the variance in the Reproducing Kernel Hilbert Space (RKHS).

ITL provides an alternative approach for clustering. Clustering methods based on ITL seek to learn the structure of the dataset from entropy and dissimilarity measures based on divergences between densities. One such method is presented in Chapter 4. For the benefit of the reader, this chapter gives a brief summary of the concept of information and entropy, before looking at standard measures of entropy and divergence. More entropy and divergence measures can be found in [21].

3.2 Information

This section is based on [30, Chapter 1]. Information theory was developed by Claude Shannon in his 1948 paper [35] "to deal with the problem of optimally transmitting messages over noisy channels" [30]. The concept of *information* was first introduced by Hartley twenty years earlier in 1928 as a measure of the unexpectedness of a message [15]. Consider a message that consists of symbols drawn from a finite set of S possible symbols. If it is binary code the only possible symbols are "0" and "1", which gives $S = 2$. If the message is n symbols long, Hartley defined the "amount of information" to be

$$H_{\mathrm{H}} = \log_{10} S^n = n \log_{10} S \tag{3.1}$$

where H_{H} is an uncertainty measure with the subscript "H" stands for Hartley. No probabilistic reasoning was used to formulate the definition in Eq. (3.1). Hartley pointed out in [15] that a logarithmic function was the most natural choice for an uncertainty measure. A more precise justification of the use of logarithms was provided by Shannon [35]. We will return to uncertainty measures in Sections 3.3 and 3.4.

While originally developed to characterize messages, the generalization to describe random events broadened the range of applications for information theory. Before providing a formal definition, let us consider an example of information related to random events. Informally we can say that knowing the outcome of the toss of a fair die provides more information than the flip of a fair coin because the outcome is harder to guess (less predictable). Also, knowing that the outcome of a toss of two dice is two and four provides the same information as the two independent pieces of information; that the toss of one die is two and the toss of another is four.

Consider a discrete random variable x with possible outcomes $S_X = \{s_1, \ldots, s_n\}$ with probabilities $P_{S_X} = \{P_1, \ldots, P_n\}$ with $P_j = P(x = s_j)$. As they are probabilities $0 \leq P_j \leq 1$ and $\sum_{j=1}^{n} P_j = 1$. The information I associated with outcome s_j is then defined as

$$I_j = \log_2 \frac{1}{P_j} = -\log_2 P_j \tag{3.2}$$

Notice that $-\log_2 P_j$ is a new random variable on S defined by the probability mass function of the random variable x [30]. Also note that in Eq. (3.2) the base of the logarithm is 2 and not 10 as in Eq. (3.1). Of course it is easy to switch between different base numbers by multiplying with a constant related to the bases. Change from base a to base b is done by multiplying by $\log_b a$. Today the base 2 logarithm is preferred as it relates to digital binary information. Then the information is measured in *bit*. If we use the base 10 logarithm in Eq. (3.2), the unit is called the *hartley*.

Another intuitive idea is that the information content of independent messages should be additive. It is easy to show that this is achieved with the formulation in Eq. (3.2). If we have observed the independent random events A and B with joint probability $P(A \cap B) = P(A|B)P(B) = P(A)P(B)$, we can denote the information provided by observing this joint event as $I(A, B)$:

$$I(A, B) = \log_2 \frac{1}{P(A)P(B)} = \log_2 \frac{1}{P(A)} + \log_2 \frac{1}{P(B)} = I(A) + I(B) \tag{3.3}$$

where $I(A)$ is the information provided by observing event A and $I(B)$ is the information provided by observing event B. So the product rule of logarithms ensures that the information of independent events is additive.

If we have perfect knowledge of a message in advance, we say that the information content of the message is zero. This is easily seen as this would correspond to $P_j = 1$ in Eq. (3.2), which would give $I_j = \log_2 1 = 0$. An unexpected message however will have high information content.

We started by defining Hartley's uncertainty measure, or "amount of information", for a set of n possible symbols, S in Eq. (3.1) and will return to uncertainty measures in the next section. Using the probabilistic reasoning described here, Shannon defined the uncertainty of the random variable x. He called his uncertainty measure *entropy*.

3.3 Shannon Entropy

Shannon wanted a measure to describe how uncertain we are of an outcome. If we have n possible outcomes with the probability of each outcome

given by $P_1, P_2, \ldots, P_n$, denote the uncertainty measure of the outcome as $H(P_1, P_2, \ldots, P_n)$. Shannon required that H should satisfy the following properties [35]:

1. H should be continuous in the P_j. This means that a small change in the probability distribution only results in a small change in the entropy.

2. If all the P_j are equal, $P_i = \frac{1}{n}$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, or uncertainty, when there are more possible events.

3. If a choice can be broken down into two successive choices, the original H should be the sum of the individual values of H weighted by the probability of that choice.

From these assumptions it is possible to specify an uncertainty measure.

Theorem 3. *Shannon entropy. The only H satisfying the three above assumptions is of the form*

$$H_S = -C \sum_{j=1}^{n} P_j \log P_j \tag{3.4}$$

where is C a positive constant and the subscript S is to emphasize that it is the Shannon entropy.

This was proven in [35]. If the probabilities are associated with the outcomes of the random variable x, we will use the more convenient notation $H_{\mathrm{S}}(x)$. The constant C merely specifies which unit of measure we use. The common choice is bits, and the Shannon entropy for x can then be written as

$$H_{\mathrm{S}}(x) = -\sum_{j=1}^{n} P_j \log_2 P_j$$

with the probabilities defined as before. We notice that in the sum, the probabilities are multiplied by the logarithm of the probabilities. This is per definition an expectation as it a function of the random variable multiplied by the probability mass function for that variable. So

$$H_{\mathrm{S}}(x) = -\sum_{j=1}^{n} P_j \log_2 P_j = \mathrm{E}\left\{\log_2 f(x)\right\} \tag{3.5}$$

where $f(x)$ is the probability mass function of the random variable x.

Shannon also extended his work in [35] to consider cases where messages are continuous (random) variables. Given a continuous distribution function, $f(x)$, we define the entropy in a similar manner to Eq. (3.5):

$$H_{\mathrm{S}}(x) = -\int_{-\infty}^{\infty} f(x) \log_2 [f(x)]\, dx$$

We see that this also can be expressed as an expectation:

$$H_{\mathrm{S}}(x) = -\int_{-\infty}^{\infty} f(x) \log_2 [f(x)]\, dx = -\mathrm{E}\left\{\log_2 f(x)\right\} \tag{3.6}$$

Let $\mathbf{x}$ be a $l \times 1$ continuous random vector with probability density function $f(\mathbf{x})$, the expression for Shannon entropy becomes

$$H_{\mathrm{S}}(\mathbf{x}) = -\int_{-\infty}^{\infty} f(\mathbf{x}) \log_2 [f(\mathbf{x})]\, d\mathbf{x} = -\mathrm{E}\left\{\log_2 f(\mathbf{x})\right\} \tag{3.7}$$

The entropy defined for continuous distributions has most of the properties it has the discrete case. One important exception is that in the continuous case, the entropy measure is relative to the coordinate system. This means that the entropy of a continuous distribution can be negative. In the discrete case the entropy measures in an absolute way the randomness of a stochastic variable and is always non negative [35].

3.4 Renyi Entropy

This section is based on [30, Chapter 2]. While the Shannon entropy occupies a central role in information theory, it is not the only meaningful uncertainty measure. Renyi entropy was developed by the Hungarian mathematician Alfred Renyi as a generalization of Shannon entropy [32]. We notice that the Shannon entropy in Eq. (3.5) can be written as

$$H_{\mathrm{S}}(x) = \sum_{j=1}^{n} P_j I_j \tag{3.8}$$

with the information I_j defined in Eq. (3.2). This formulation uses a linear averaging operation. In the theory of means, any monotonic function $g(\cdot)$ with an inverse $g^{-1}(\cdot)$ can be used to define a general mean. The general mean associated with $g(\cdot)$ for the information can then be defined as

$$H(x) = g^{-1}\left(\sum_{j=1}^{n} P_j g(I_j)\right) \tag{3.9}$$

where $g(\cdot)$ is a Kolmogorov-Nagumo invertible function [30], and is the so called quasi-linear mean. Renyi proved that when one requires that the postulate of additivity for independent events should hold, it restricts $g(\cdot)$ to two possible classes. One is, expressed in terms of the variable r, $g(r) = Cr$ where C is a constant. This reduces the quasi-linear mean to the ordinary mean and yields the Shannon entropy [30]. The other functional class, written as a function of the variable r, is

$$g(r) = C\, 2^{(1-\alpha)r} \tag{3.10}$$

which is used to define the Renyi entropy of order α for a discrete random variable x with probabilities P_j as

$$H_\alpha(x) = \frac{1}{1-\alpha} \log_2 \left[\sum_{j=1}^{n} P_j^\alpha \right] \tag{3.11}$$

with $\alpha \geq 0$ and $\alpha \neq 1$. Compared to the Shannon entropy in Eq. (3.5) we notice that the logarithm is now placed outside the summation. So the probability mass function (pmf) weights the $\alpha - 1$ power of the pmf as $\sum P_j^\alpha = \sum P_j P_j^{\alpha-1}$, and the logarithm is taken of this expression. Using this it is easy to see that we also can express Eq. (3.11) in terms of an expectation

$$H_\alpha(x) = \frac{1}{1-\alpha} \log_2 \left[\sum_{j=1}^{n} P_j^\alpha \right] = \frac{1}{1-\alpha} \log_2 \mathrm{E}\left\{ f^{\alpha-1}(x) \right\} \tag{3.12}$$

where $f(x)$ is the pmf of x, which is defined by the probabilities $P_1, \ldots, P_n$.

The α parameter provides more flexibility as it allows several measures of uncertainty within a given distribution. Considering $H_\alpha(x)$ as a function of α is called the spectrum of Renyi information and plotting it is useful in statistical inference [37].

Various axiomatizations of Renyi and Shannon entropy exists, in [30] a comparison between axioms for both types are presented. As mentioned, Renyi entropy is a generalization of Shannon entropy and it can be shown by using the l'Hôpital rule that $\lim_{\alpha \to 1} H_\alpha(x) = H_\mathrm{S}(x)$ and thus both can be included in a single unifying axiomatic [30].

In [30] the argument of the logarithm is called the α *information potential* and denoted as

$$V_\alpha(x) = \sum_{j=1}^{n} P_j^\alpha = \mathrm{E}\left\{ f^{\alpha-1}(x) \right\} \tag{3.13}$$

and Eq. (3.12) can be rewritten as

$$H_\alpha(x) = \frac{1}{1-\alpha}\log_2\left[V_\alpha(x)\right] \tag{3.14}$$

As for Shannon entropy, Renyi entropy can be defined for continuous random vectors in the same way as in Eq. (3.7). The Renyi entropy of order α is then

$$H_\alpha(\mathbf{x}) = \frac{1}{1-\alpha}\log_2\left[\int_{-\infty}^{\infty} f^\alpha(\mathbf{x})d\mathbf{x}\right] = \frac{1}{1-\alpha}\log_2 \mathrm{E}\left\{f^{\alpha-1}(\mathbf{x})\right\} \tag{3.15}$$

and also

$$H_\alpha(\mathbf{x}) = \frac{1}{1-\alpha}\log_2\left[V_\alpha(\mathbf{x})\right] \tag{3.16}$$

where $V_\alpha(\mathbf{x}) = \log_2 \mathrm{E}\left\{f^{\alpha-1}(\mathbf{x})\right\}$ is the information potential.

From an engineering perspective, one must often estimate entropy from data, and one should consider this when choosing an uncertainty measure for entropy-based algorithms. In this respect, Renyi entropy represents a significant improvement over Shannon entropy as it is no longer necessary to use a parametric estimator for the pdf. This is a result of the logarithm appearing outside the sum or integral which allows the use of kernel density estimation (Parzen windowing) because the estimate for the information potential V_α can be expressed in terms of pairwise interactions between the data points.

While this is possible for all $\alpha \neq 1$, it is perhaps best illustrated when estimating Renyi's quadratic entropy, $H_2(\mathbf{x}) = -\log_2\left[\int f^2(\mathbf{x})d\mathbf{x}\right] = -\log_2 \mathrm{E}\left\{f(\mathbf{x})\right\}$. Then the estimated entropy can be expressed in terms of a double sum of a kernel function of pairs of data points. ITL is based on optimizing expressions related to Renyi's quadratic entropy. This is done in Section 4.2 is the foundation[2] of the clustering method presented in Chapter 4.

3.5 Entropy in Physics

In physics, the concept of entropy has several interpretations, one is as a descriptor of the disorder of a physical system in thermodynamics [13]. The form of Eq. (3.4) is the entropy as defined in certain formulations of statistical mechanics [35]. To provide some intuition for what entropy is, we will briefly look at how it is defined in physics.

[2]The spectral clustering method presented there only works when using the Renyi entropy of order $\alpha = 2$.

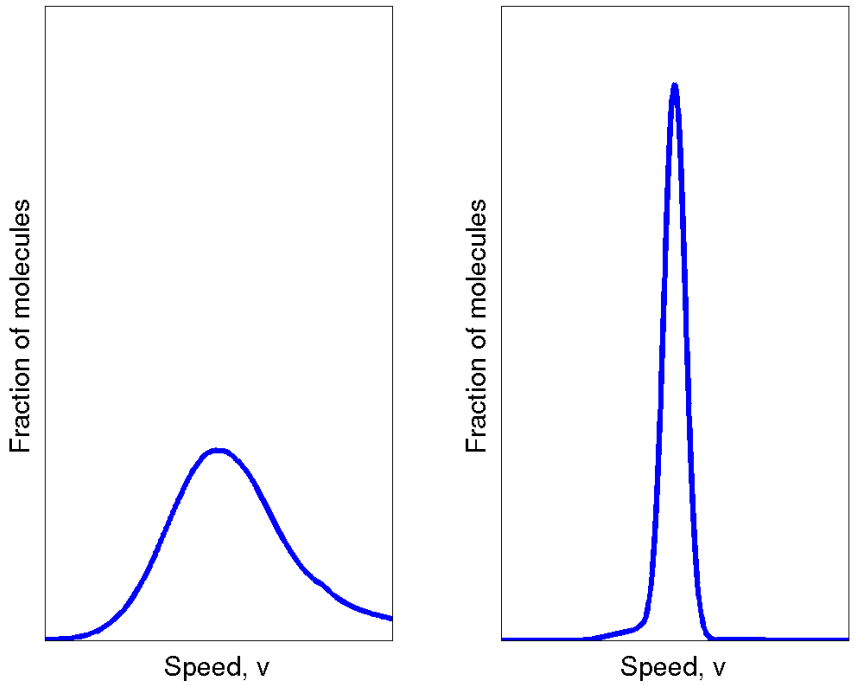


Figure 3.1: Speed of molecules in a gas.

The *microstate* of a system is for each particle to specify the position and velocity. Generally there are far too many particles to specify the microstate, and one instead specifies the system by its macroscopic properties such as temperature, number of particles, pressure and so on. These properties specifies a system's *macrostate*. However, many different microstates can correspond to the same macrostate [13]. In statistical thermodynamics, entropy is used to say something about the uncertainty which remains about the state of a system once all the macroscopic properties have been specified [17].

Let us consider a simple example with two identical rooms filled with the exact same number of gas molecules, where the positions of the molecules relative to each other and to the room are known and identical. We also know the direction of the movement for each particle, but not the speed.

Figure 3.1 is based on Figure 15-19 in [13] and is meant to illustrate the distribution of molecule speeds in the gas of each room. The rightmost plot shows an orderly distribution, in which all molecules have nearly the same speed. The leftmost plot shows a gas with a more disorderly distribution of

speeds, hence this system has higher entropy (of course assuming the systems are equal in all other respects).

As mentioned, entropy is an uncertainty measure for the system. An intuition for this is that if we did not know which speed belonged to which molecule, we would get a better estimate of the position and velocity of the molecules after they move from their original positions if the speeds were distributed according to the rightmost plot.

Entropy in the information theoretic setting is similar in that it describes the disorder, or uncertainty, of a random variable. If the plots in Fig. 3.1 were probability density functions for random variables, the conclusion would be the same as for physical entropy, the random variable belonging to the leftmost plot has higher entropy.

3.6 Divergence

Divergence is a measure of statistical similarity and can be thought of as a generalization of algebraic distance measures to probability distribution spaces [8]. While always non-negative, divergences generally do not satisfy the conditions for a metric and hence can not be called distance functions [30].

We start by defining the well known Kullback-Leibler divergence (KLD) which also illustrates the difference between divergences and distances. The Kullback-Leibler divergence D_{KL} between the two multivariate probability density functions $f(\mathbf{x})$ and $q(\mathbf{x})$ is defined as:

$$D_{\mathrm{KL}}(f||q) = \int_{-\infty}^{\infty} f(\mathbf{x}) \log \left[\frac{f(\mathbf{x})}{q(\mathbf{x})} \right] d\mathbf{x} = \mathrm{E}_f \left\{ \log \left[\frac{f(\mathbf{x})}{q(\mathbf{x})} \right] \right\} \tag{3.17}$$

where E_f is the expectation with respect to the distribution $f(\mathbf{x})$. We see from Eq. (3.17) that it does not satisfy the symmetry property of metrics as $D_{\mathrm{KL}}(f||q)$ differs from $D_{\mathrm{KL}}(q||f)$ in general. Nor does KLD obey the triangle inequality.

While these properties may be useful in some applications, one often wants to work with symmetric dissimilarity measures [30]. It is possible to define a symmetric divergence measure known as Jeffrey divergence in terms of the Kullback-Leibler divergence:

$$D_{\mathrm{J}}(f||q) = \sqrt{\frac{1}{2} \left(D_{\mathrm{KL}}(f||q) \right)^2 + \frac{1}{2} \left(D_{\mathrm{KL}}(q||f) \right)^2} \tag{3.18}$$

with D_{KL} defined in Eq. (3.17).

Another symmetric measure is the Cauchy-Schwarz (CS) divergence, D_{CS} where

$$D_{\text{CS}}(f||q) = -\log \frac{\int f(\mathbf{x})q(\mathbf{x})d\mathbf{x}}{\sqrt{\int f(\mathbf{x})^2 d\mathbf{x} \int q(\mathbf{x})^2 d\mathbf{x}}} \tag{3.19}$$

where $f(\mathbf{x})$ and $q(\mathbf{x})$ are pdfs as before and the limits of the integral are $-\infty$ and ∞. Note that it is possible to rewrite Eq. (3.19) and Eq. (3.17) by using the quotient property of logarithms. For the CS divergence this gives us the equivalent expression

$$D_{\text{CS}}(f||q) = -\frac{1}{2}\log \frac{\left(\int f(\mathbf{x})q(\mathbf{x})d\mathbf{x}\right)^2}{\int f(\mathbf{x})^2 d\mathbf{x} \int q(\mathbf{x})^2 d\mathbf{x}} \tag{3.20}$$

It is easy to see from Eq. (3.19) that it is symmetric, $D_{\text{CS}}(f||q) = D_{\text{CS}}(q||f)$, and hence one can use the notation $D_{\text{CS}}(f, q)$ to indicate the symmetry.

Whereas the form of the KL divergence resembles Shannon entropy, CS divergence is associated with the Renyi entropy of order $\alpha = 2$. This means that the estimated CS divergence can be expressed in terms of a double sum by using KDE in the same way as Renyi's quadratic entropy.

More divergence measures and their properties can be found in [30].

Chapter 4

Kernel Entropy Component Analysis

4.1 Introduction

A new method for data transformation and dimensionality reduction based on preserving the estimated Renyi entropy of order $\alpha = 2$ in the dataset was presented in [18]. The reason for using this particular entropy estimate is that when using kernel density estimation (KDE) it can be expressed in terms of the kernel matrix. Then a transformation that preserves the estimated entropy can be found by using the eigenvectors of this matrix. This makes it a spectral clustering method. The method was named Kernel Entropy Component Analysis (KECA).

While the starting point of the data transformation is different from the one for Kernel Principal Component (KPCA) in Section 2.5.3, the resulting optimization problem can be seen as projecting onto a subset of unentered Kernel Principal Component (KPCA) axes if the kernel used satisfies Theorem 1 (Mercer's Theorem). The resulting data transformation may be the same or strikingly different, depending on the data and the parameters chosen. It will always be different from regular, centered, KPCA.

A clustering method based on KECA, similar to clustering for KPCA, was also proposed. The clustering method seeks to exploit the distinct angular structure produced by the KECA transformation [18].

4.2 Estimating Renyi's Quadratic Entropy

As seen in Chapter 3, there is more than one way to define entropy. Kernel Entropy Component Analysis (KECA) is based on preserving the estimated

Renyi quadratic entropy, H_2. Given the $l \times 1$ stochastic vector $\mathbf{x}$ with probability density function $f(\mathbf{x})$, using equation Eq. (3.15) with $\alpha = 2$ gives the expression

$$H_2(f) = -\log\left[\int_{-\infty}^{\infty} f^2(\mathbf{x})d\mathbf{x}\right] = -\log\left[V_2(\mathbf{x})\right] \tag{4.1}$$

where $V_2(\mathbf{x})$ is the information potential of the the probability density function $f(\mathbf{x})$. The subscript denoting the base of the logarithm as 2 has been skipped for notational convenience and to avoid confusion with the subscript on the entropy. The information potential can also be formulated as

$$V_2(\mathbf{x}) = \mathrm{E}_f\left\{f(\mathbf{x})\right\} \tag{4.2}$$

where $\mathrm{E}_f\left\{\cdot\right\}$ is the expectation with respect to the density $f(\mathbf{x})$. Because it can be written as an expectation, the information potential can be estimated as the mean of the density. We need to estimate the density to obtain an estimate of the information potential. Once we have this estimate, the estimate of the entropy is found by simply taking the logarithm and multiplying by -1 as we see from Eq. (4.1). Hence we will focus on the information potential.

We may use Kernel Density Estimation, also called Parzen Windowing, described in Section 2.4.1 where Eq. (2.50) gives the pdf estimate for $\mathbf{x}$ given the data $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$

$$\hat{f}(\mathbf{x}) = \frac{1}{n}\sum_{j=1}^{n} K(\mathbf{x}, \mathbf{x}_j) \tag{4.3}$$

where $K(\mathbf{x}, \mathbf{x}_j)$ is the kernel centered at $\mathbf{x}_j$, also called the Parzen window. In order for $\hat{f}(\mathbf{x})$ to be a valid density estimate, the kernel $K(\mathbf{x}, \cdot)$ must be a density function itself. We can now estimate the information potential as

$$\hat{V}_2(\mathbf{x}) = \hat{\mathrm{E}}_f\left\{f(\mathbf{x})\right\} = \frac{1}{n}\sum_{i=1}^{n}\hat{f}(\mathbf{x}_i) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{n}\sum_{j=1}^{n} K(\mathbf{x}_i, \mathbf{x}_j) \tag{4.4}$$

It was pointed out in [14] that Eq. (4.4) can be formulated in terms of the elements of $n \times n$ kernel matrix $\mathbf{K}$:

$$\hat{V}_2(\mathbf{x}) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} K(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{n^2}\mathbf{1}^T\mathbf{K}\mathbf{1} \tag{4.5}$$

where $K(\mathbf{x}_i, \mathbf{x}_j)$ is element (i, j) of the kernel matrix $(K_{i,j})$, and $\mathbf{1}$ is the $n \times 1$ one-vector with all elements equal to one.

We recall from Section 2.5.3 that because $\mathbf{K}$ is symmetric, it may be eigendecomposed as $\mathbf{K} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T$, where $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues $\lambda_1, \ldots, \lambda_n$ of $\mathbf{K}$ and $\mathbf{E}$ is a matrix with the corresponding eigenvectors $\mathbf{e}_1, \ldots, \mathbf{e}_n$ as columns.

This means that we can rewrite Eq. (4.4) in terms of the eigenvalues and eigenvectors

$$\hat{V}_2(\mathbf{x}) = \frac{1}{n^2} \sum_{i=1}^{n} \left(\sqrt{\lambda_i} \mathbf{e}_i^T \mathbf{1} \right)^2 \tag{4.6}$$

Equation (4.6) shows that the Renyi entropy estimator is composed of projection onto all the uncentered kernel PCA axes, provided that the kernel function used for Parzen Windowing is positive semi definite [18]. Notice that only if $\lambda_i \neq 0$ *and* $\mathbf{e}_i^T \mathbf{1} \neq 0$ does the projection onto the i'th principal axis contribute to the overall entropy estimate.

We will now use the expression in Eq. (4.6) to define a transformation onto a subset of these axes. As seen from Eq. (4.1), preserving the estimated entropy is equivalent to preserving the estimated information potential.

If we assume that the kernel used for Parzen windowing is positive semi definite (psd), we can interpret the Renyi entropy estimator in terms of projections $\mathbf{x} \in \mathbb{X} \mapsto \boldsymbol{\phi}(\mathbf{x}) \in H$ in a Reproducing Kernel Hilbert Space H as defined in Section 2.5.2.

4.3 The KECA Transformation

Assume that we are given the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ of $l \times 1$ data vectors. Now we want to define a transformation, $\mathbf{x}_j \in \mathbb{R}^l \mapsto \mathbf{y}_j \in \mathbb{R}^s$, $j = 1, \ldots, n$, obtained by projecting onto a subspace spanned by the s uncentered KPCA axes that contributes the most to the estimated quadratic Renyi entropy of the data [18].

If we organize the transformed dataset $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ into a matrix

$$\mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \vdots \\ \mathbf{y}_n^T \end{bmatrix}$$

where $\mathbf{y}_j$, $j = 1, \ldots, n$ are $s \times 1$ vectors. A projection onto s kernel PCA axes can be expressed as

$$\mathbf{Y} = \mathbf{\Lambda}_s^{\frac{1}{2}} \mathbf{E}_s^T \tag{4.7}$$

where $\mathbf{\Lambda}_s$ is a $s \times s$ diagonal matrix with eigenvalues and $\mathbf{E}_s$ is a $n \times s$ matrix with the s corresponding eigenvectors as columns. In centered kernel PCA,

these are the s largest eigenvalues, as this will be the axes that contribute the most to the covariance in the Reproducing Kernel Hilbert Space (RKHS). However, here we wish to preserve the axes that contribute the most to the estimated information potential in the *input space*, and hence the estimated entropy.

Let us denote the estimated information potential based on s KPCA axes as $\hat{V}_2^*(\mathbf{x})$. This information potential is then expressed by [18]:

$$\hat{V}_2^*(\mathbf{x}) = \frac{1}{n^2}\mathbf{1}^T\mathbf{E}_s\mathbf{\Lambda}_s\mathbf{E}_s^T\mathbf{1} = \frac{1}{n^2}\mathbf{1}^T\mathbf{K}_{\text{eca}}\mathbf{1} \tag{4.8}$$

with $\mathbf{K}_{\text{eca}} = \mathbf{E}_s\mathbf{\Lambda}_s\mathbf{E}_s^T$ and $\mathbf{1}$ is a $n \times 1$ column vector with all elements equal to one. We now find *which KPCA axes* we want to project the data onto as the axes that *minimize the difference* in estimated quadratic information potential when $s < n$

$$\mathbf{Y} = \mathbf{\Lambda}_s^{\frac{1}{2}}\mathbf{E}_s^T \; : \min_{\lambda_1,\mathbf{e}_1,\ldots,\lambda_n,\mathbf{e}_n} \hat{V}_2(\mathbf{x}) - \hat{V}_2^*(\mathbf{x}) \tag{4.9}$$

where $\hat{V}_2(\mathbf{x})$ is defined in Eq. (4.5). By inserting this, and the expression in Eq. (4.8) we get

$$\mathbf{Y} = \mathbf{\Lambda}_s^{\frac{1}{2}}\mathbf{E}_s^T \; : \min_{\lambda_1,\mathbf{e}_1,\ldots,\lambda_n,\mathbf{e}_n} \mathbf{1}^T\mathbf{K}\mathbf{1} - \mathbf{1}^T\mathbf{K}_{\text{eca}}\mathbf{1} \tag{4.10}$$

We now use Eq. (4.6) to write the minimization problem in terms of a sum

$$\mathbf{Y} = \mathbf{\Lambda}_s^{\frac{1}{2}}\mathbf{E}_s^T : \min_{\lambda_1,\mathbf{e}_1,\ldots,\lambda_n,\mathbf{e}_n} \frac{1}{n^2}\sum_{i=1}^{n}\left(\sqrt{\lambda_i}\mathbf{e}_i^T\mathbf{1}\right)^2 - \frac{1}{n^2}\sum_{i=1}^{s}\left(\sqrt{\lambda_i}\mathbf{e}_i^T\mathbf{1}\right)^2 \tag{4.11}$$

It is easy to see that the minimum of Eq. (4.11) is given by

$$\min_{\lambda_1,\mathbf{e}_1,\ldots,\lambda_n,\mathbf{e}_n} \frac{1}{n^2}\mathbf{1}^T\left(\mathbf{K} - \mathbf{K}_{\text{eca}}\right)\mathbf{1} = \frac{1}{n^2}\sum_{i=s+1}^{n}\left(\sqrt{\lambda_i}\mathbf{e}_i^T\mathbf{1}\right)^2 \tag{4.12}$$

where

$$\left(\sqrt{\lambda_j}\mathbf{e}_j^T\mathbf{1}\right)^2 \geq \left(\sqrt{\lambda_i}\mathbf{e}_i^T\mathbf{1}\right)^2 \;\forall\; j = 1,\ldots,s\,,\; i = s+1,\ldots,n \tag{4.13}$$

Then we can define the KECA transform for the dataset $\mathbb{X}$:

- Construct the kernel matrix $\mathbf{K}$ with elements $K_{i,j} = K(\mathbf{x}_i,\mathbf{x}_j)$, $i,j = 1,2\ldots,n$ where $K(\cdot,\cdot)$ is a kernel function satisfying the requirements of kernel density estimation (K must itself be a density function).

- Perform the eigendecomposition $\mathbf{K} = \mathbf{E}\mathbf{\Lambda}\mathbf{E}^T$.
- Order the eigenvalue-eigenvector pairs such that $\psi_1 \geq \psi_2 \geq \ldots \geq \psi_n$, where $\psi_j = \left(\sqrt{\lambda_j}\mathbf{e}_j^T\mathbf{1}\right)^2$, $j = 1, \ldots n$.
- Define $\mathbf{\Lambda}_s$ as the $s \times s$ matrix with the s first eigenvalues on the diagonal and $\mathbf{E}_s$ as the $n \times s$ matrix with the corresponding eigenvectors as columns.
- The transformed dataset $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n\}$ is then given as the columns of the matrix $\mathbf{Y} = \mathbf{\Lambda}_s^{\frac{1}{2}}\mathbf{E}_s^T$.

Note that the above procedure is valid even if the kernel used for density estimation does not satisfy Theorem 1, but then the intuition on projecting onto a subset of uncentered KPCA axes no longer applies.

4.4 KECA Spectral Clustering

Kernel ECA can be used for a spectral clustering method in the same way as KPCA. The dataset is projected onto a user defined number of axes, where the axes are the ones minimizing the expression in Eq. (4.11). These are found as described above. Then a simple clustering algorithm, often k-means, is used on the transformed dataset. It was pointed out in [18] that the transformed dataset tends to have a distinct angular structure *"where clusters are distributed more or less in different angular directions with respect to the origin of the kernel feature space"*. It was therefore suggested that one takes advantage of this by clustering based on a cost function capable of capturing this angular structure.

In [19] it was shown that the Cauchy-Schwarz (CS) divergence in Eq. (3.19) corresponds to a measure of the cosine of the angle between the mean vectors in the kernel feature space. The CS divergence between the pdf of the ith cluster $f_i(\mathbf{x})$ and the overall data pdf $f(\mathbf{x})$ is given by $D_{\mathrm{CS}}(f_i, f) = -\log\left[V_{\mathrm{CS}}(f_i, f)\right]$ where

$$V_{\mathrm{CS}}(f_i, f) = \frac{\int f_i(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}}{\sqrt{\int f_i(\mathbf{x})^2 d\mathbf{x} \int f(\mathbf{x})^2 d\mathbf{x}}} \tag{4.14}$$

Let us assume that the n_i data points $\mathbf{x}_l \in \mathbb{X}_i$ are generated from $f_i(\mathbf{x})$ corresponding to the cluster C_i. This is a subset of the n data points $\mathbf{x}_t \in \mathbb{X}$ generated from the overall data pdf $f(\mathbf{x})$. It was shown in [19] that estimating $V_{\mathrm{CS}}(f_i, f)$ by Parzen windowing gives

$$\hat{V}_{\mathrm{CS}}(f_i, f) = \cos \angle(\mathbf{m}_i, \mathbf{m}) \tag{4.15}$$

Where $\mathbf{m}_i = \frac{1}{n_i} \sum_{\mathbf{x}_l \in \mathbb{X}_i} \boldsymbol{\phi}(\mathbf{x}_l)$ is mean vector of cluster C_i in the kernel feature space, and $\mathbf{m} = \frac{1}{n} \sum_{\mathbf{x}_t \in \mathbb{X}} \boldsymbol{\phi}(\mathbf{x}_t)$ is overall kernel space mean of the data. The Cauchy-Schwarz divergence is basically a measure of of the cosine of the angle between these mean vectors [18].

A reasonable choice for an angle based clustering cost function can then be defined in terms of the kernel feature space dataset as

$$J(C_1, C_2, \ldots, C_k) = \sum_{i=1}^{k} \cos \angle(\mathbf{m}_i, \mathbf{m}) \tag{4.16}$$

It was shown theoretically in [19] that this corresponds to the *kernel* k-means clustering objective using an angular distance measure. The kernel k-means algorithm uses kernel functions are used to calculate distances in the RKHS, rather than calculating distances in the input space.

We do the optimization of this angle-based cost function with respect to cluster allocation using the KECA transformed data $\mathbf{y}_j$ to represent the feature space dataset $\boldsymbol{\phi}(\mathbf{x}_j)$, $j = 1, \ldots, n$ because of its angular structure. Then the optimization procedure is then simply the "regular" k-means algorithm using angular distances in the KECA space [18]. K-means is guaranteed to converge to a local optimum.

In [27] it was suggested that one projects the data onto as many axes as the number of clusters one wants in the result when using spectral clustering. This implies that when performing the transformation to the KECA space, $\mathbf{x}_j \in \mathbb{R}^l \mapsto \mathbf{y}_j \in \mathbb{R}^s$, $j = 1, \ldots, n$, s is selected to be equal to the number of clusters. This is motivated by the ideal case when the clusters are easily separable, much like the *connected components* of a graph discussed in Section 2.6.2. While the kernel matrix defined here would be characterized as fully connected according to the graph theory presented in Section 2.6.2, the kernel size can be so small that the matrix is essentially block diagonal. Then some of the eigenvectors of $\mathbf{K}$ will be similar to the indicator vectors for a particular cluster, as defined for connected components in Theorem 2.

The clustering method can then be summarized in the following steps:

1. Obtain $\mathbb{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$ from the KECA transform as the columns of $\mathbf{Y} = \boldsymbol{\Lambda}_s^{\frac{1}{2}} \mathbf{E}_s^T$.

2. Initialize means $\mathbf{m}_i$, $i = 1, \ldots, k$ for the k-means algorithm.

3. For all t: $\mathbf{x}_t \to C_i$: $\max_i \cos \angle(\mathbf{y}_t, \mathbf{m}_t)$.

4. Update mean vectors.

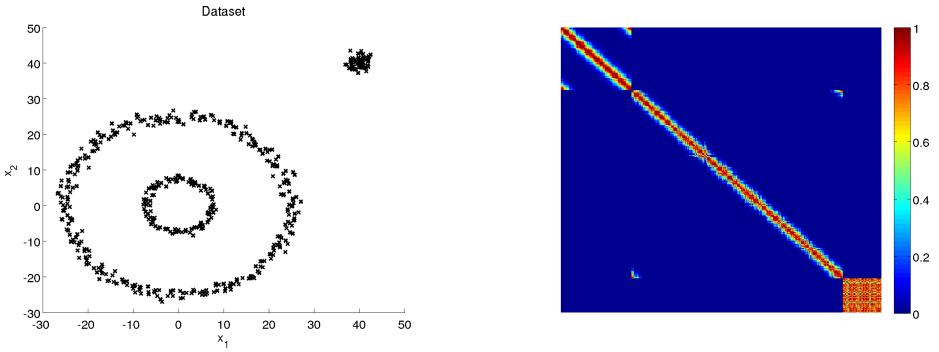
5. Repeat steps 3 and 4 until convergence.

As mentioned in Section 2.2, the k-means clustering result depends on how the cluster representatives are initialized. A method for initializing the centers that exploited the angular structure and produced good results was presented in [18]. The means in step 2 are initialized as $\mathbf{m}_1 = \mathbf{y}_t$ and $\mathbf{m}_2 = \mathbf{y}_{t'}$ where the transformed data points $\mathbf{y}_t, \mathbf{y}_{t'}$ have the property

$$\cos \angle(\mathbf{y}_t, \mathbf{y}_{t'}) = \min_{j,j'} \cos \angle(\mathbf{y}_j, \mathbf{y}_{j'}) \ , \ j = 1, \ldots, n \tag{4.17}$$

Furthermore, $\mathbf{m}_i \ , \ i > 2$ is initialized as $\mathbf{y}_{t''}$ such that it minimizes $\sum_{j=1}^{i-1} \cos \angle(\mathbf{m}_j, \mathbf{y}_{t''})$.

We now use the KECA clustering method on a dataset containing two concentric rings and a points from multinormal density linearly separable from the two rings.



(a) Dataset. (b) Image plot of kernel matrix.

Figure 4.1: Two rings and a dense cluster.

The dataset without labels is shown in Fig. 4.1a. The dataset consists of 110 points in the inner ring, 330 in the outer and the dense cluster outside consists of 60 data points. The kernel matrix is calculated using a Gaussian kernel with as defined in Eq. (2.93) with bandwidth parameter $h = 3.7$. It is plotted as an image in Fig. 4.1b. Each element is between 0 and 1 and the colour red indicates a high value. As seen from Eq. (2.93), a high value means that the points are close in Euclidean distance.

Because each of the rings and the dense cluster are generated separately before being combined into the overall dataset seen in Fig. 4.1a, we know which data points belong to which cluster. The first 110 points belongs to the inner ring, the next 330 points the outer ring and the last 60 points stems from the dense cluster. Then we also know that the top 110×110 submatrix in the upper left corner of Fig. 4.1b contains the kernel affinity

measures between all pairs of points in the inner circle. Likewise, the 330×330 submatrix in middle corresponds to the outer circle and the 60×60 square in the bottom right corner corresponds to the dense cluster.

The structure seen in the submatrices corresponding to the two rings are caused by the way the rings are generated. Each ring is generated as a circle with different multivariate normal noise. This means that without the noise, data point i in one of the circles would have data point $i-1$ and $i+1$ as its two closest neighbours. This might not be true after adding the noise. However, since the noise is independent and identically multivariate normal distributed with the zero vector as mean, we would expect this to be true for many points. This explains why the elements along the diagonal are dominant. The large elements off the diagonal is from the circle overlapping, that is, element 1 is neighbours with element 2 and element 110.

We notice that the elements in the bottom right corner of Fig. 4.1b corresponding to the dense cluster has no apparent structure, as would be expected from independent noise samples.

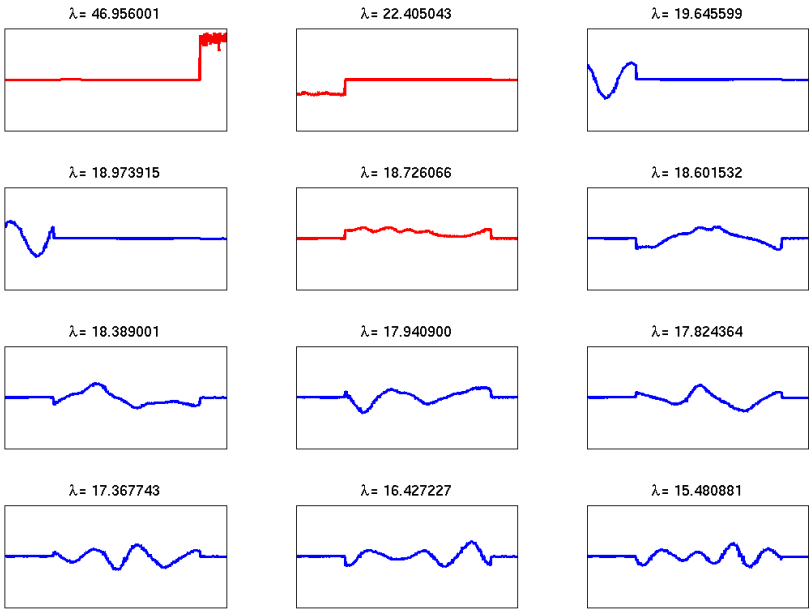


Figure 4.2: Eigenvectors.

Figure 4.2 plots the eigenvectors corresponding to the 12 largest eigenvalues of the kernel matrix in Fig. 4.1b. The eigenvectors plotted in red are the ones used by the KECA data transform according to Eq. (4.13). We notice

that the transformation has selected the eigenvectors corresponding to the largest, second and fifth largest eigenvalues.

Each element in the eigenvectors correspond to a data point in the input space. The order is the same as for the original data points, so $\mathbf{e}_i(j)$ corresponds to data point j. The elements of the eigenvector in the top left corner are approximately zero for the the two rings, and have a fairly constant positive value for the dense cluster. Hence it can be seen as a sort of indicator vector, where a large absolute[1] value in the vector indicates that the corresponding data point belongs to a particular cluster, in this case the dense cluster.

The first 110 elements of eigenvector corresponding to the second largest eigenvalue $\lambda = 22.405$ are negative, while the rest are close to zero. Hence this is an indicator vector for the inner ring. Likewise, the third and fourth eigenvectors are non-zero only for the inner ring, but these elements oscillates around zero. Then $\left(\sqrt{\lambda_3}\mathbf{e}_3^T\mathbf{1}\right)^2 \approx \left(\sqrt{\lambda_4}\mathbf{e}_4^T\mathbf{1}\right)^2 \approx 0$ and hence their contribution to the entropy estimate will be negligible, despite the large eigenvalues. Thus the last of the three eigenvector selected by kernel ECA is the one corresponding to the fifth larges eigenvalue. While the structure is not as clear as for the first two eigenvectors, we see that the eigenvector is positive only for the elements corresponding to the outer ring.

Figure 4.3 shows the result of KECA clustering method using the cosine distance for the k-means algorithm. The data KECA transform was based on the eigenvectors plotted in red in Fig. 4.2. The clustering method has managed to separate the two circles and the dense cluster in the upper right corner. There are no errors in the clustering compared to the true labels.

The data transformation based on the KECA is shown in result in Fig. 4.4a. We notice the distinct angular structure in the transformed data, and that the cosine distance appears to be a valid distance measure for the transformed data. This is as expected when we look at Fig. 4.2 since each of the eigenvectors used indicate a different cluster.

Figure 4.4b is the transformation based on using the eigenvectors corresponding to the three larges eigenvalues, plotted in the top row of Fig. 4.2. This is the result of using KPCA if one omits the centering. The outer ring is mapped to the origin since non of the first three eigenvectors have non-zero elements corresponding to this cluster. Using the third eigenvector instead of the fifth causes the blue cluster to be spread out, since both $\mathbf{e}_2$ and $\mathbf{e}_3$

[1] The sign on the indicator vector is arbitrary; if $\mathbf{e}_i$ is a solution to eigenvalue-eigenvector problem $\mathbf{K}\mathbf{e}_i = \lambda_i\mathbf{e}_i$ then so is $-\mathbf{e}_i$. The elements corresponding to the other clusters may be non-zero, but as seen in Fig. 4.2, they will be small in magnitude compared to the elements that indicate membership. One should keep this in mind if clustering based on thresholding eigenvectors.

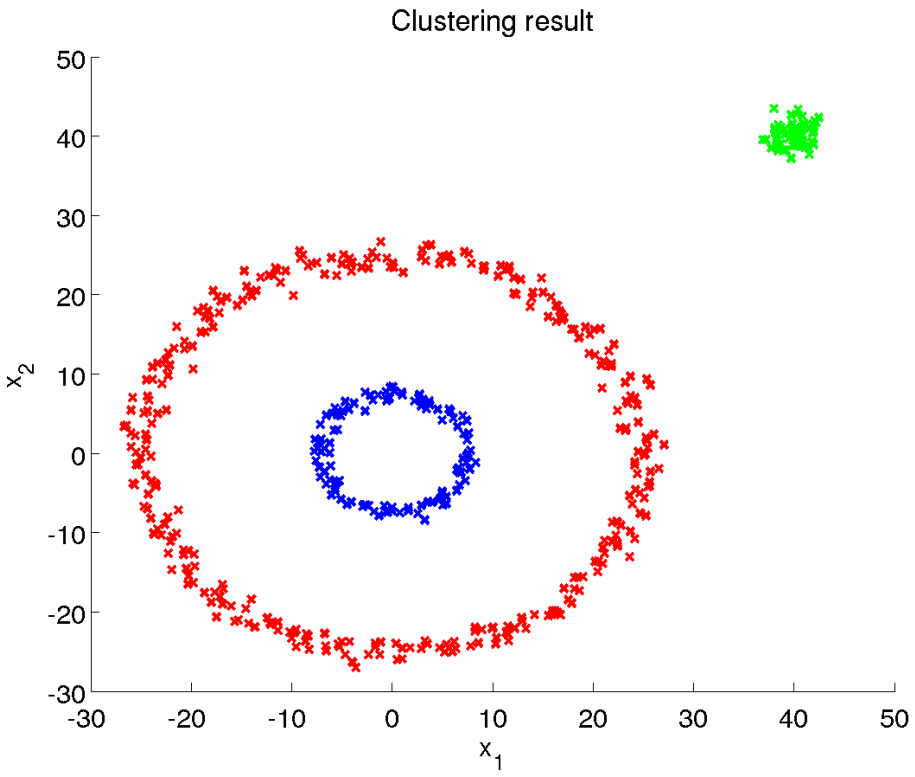


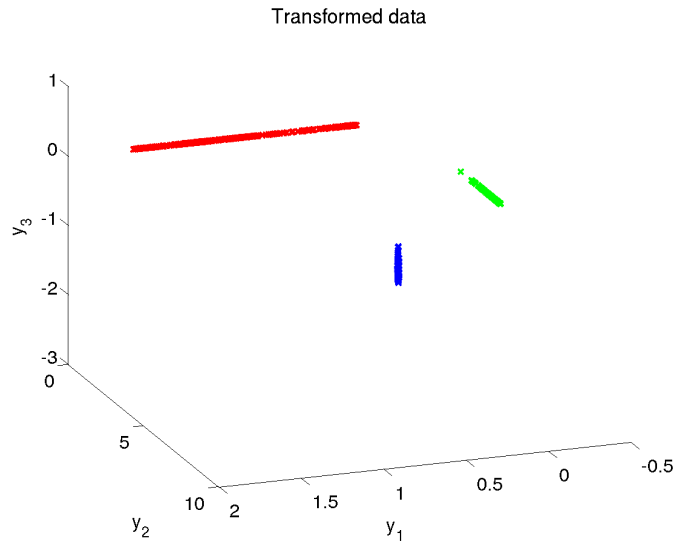
Figure 4.3: Clustering result.

have non-zero elements corresponding to the inner ring.

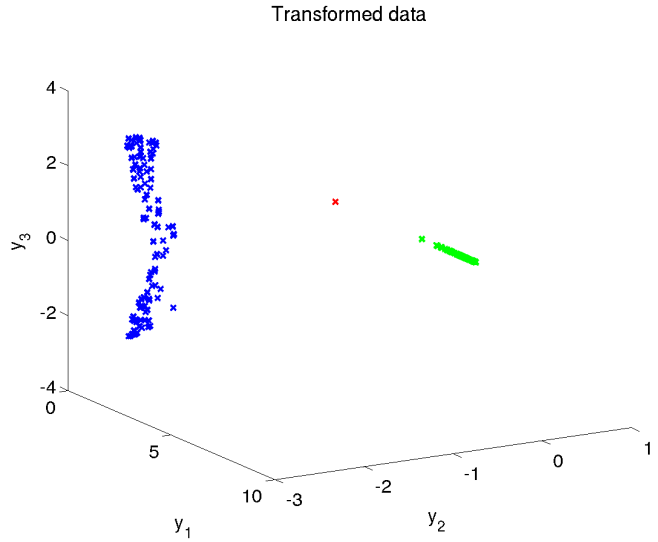
One can say that this transformation gives a two dimensional representation of the inner circle, a one dimensional representation of the dense cluster and the outer circle has collapsed onto a single point. This is why there is no distinct angular structure in this transformation, contrary to Fig. 4.4a where each cluster is represented by a line (one dimension) that is orthogonal to the other two. Thus the cosine distance measure is not a good choice for the transformed dataset in (b), and the k-means algorithm is not able to separate between the clusters using this distance measure. Using the standard squared Euclidean distance however will give an error free clustering result[2].

The transformation using standard, centered, kernel PCA is shown in Fig. 4.4c. It has similar structure to (b), but the red class corresponding to the outer circle is more spread out. Note that the scale of the axes for the centered KPCA transform is much smaller than for the other two because of the normalization of the eigenvectors as defined in Eq. (2.89). This has been

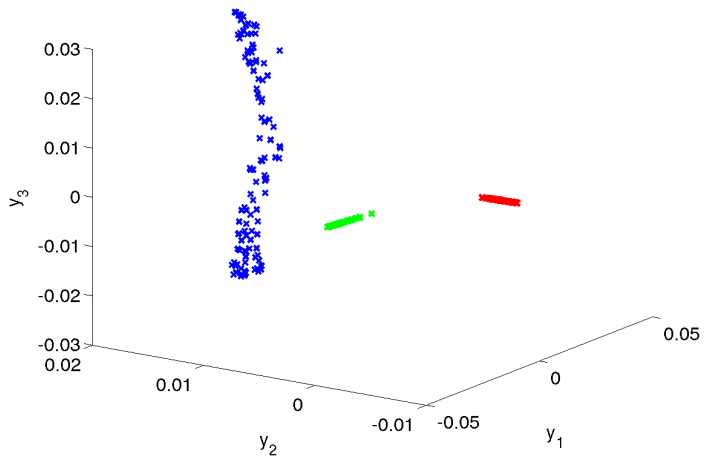
[2]For both distance measures the k-means algorithm was run 500 times with random initialization and the result with fewest errors compared to the original labels was used.



(a) KECA.



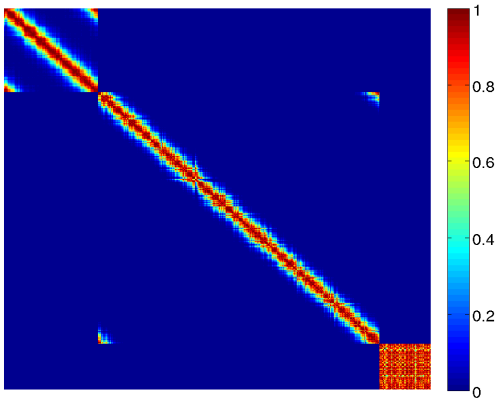
(b) Uncentered KPCA.



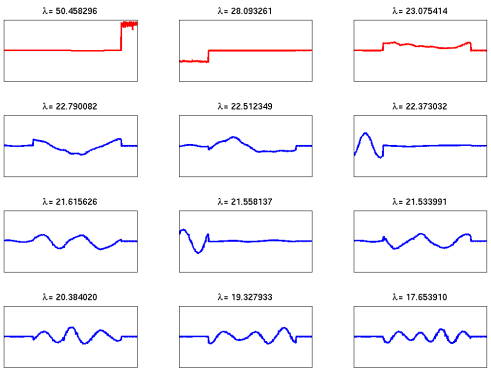
(c) Centered KPCA.

Figure 4.4: Transformed data.

omitted for the uncentered KPCA transformation for easier comparison with KECA, but the structure seen in (b) would be the same. The centered kernel matrix has different eigenvectors than the uncentered ones shown in Fig. 4.2. Note that the perspective of the plots in Fig. 4.4 are different to best show the structure of each transformation.



(a) Kernel matrix.



(b) Eigenvectors.

Figure 4.5: Increasing bandwidth to $h = 4.5$.

Figure 4.5a shows the kernel matrix for the dataset in Fig. 4.1a using a Gaussian kernel with larger bandwidth, $h = 4.5$. It does not appear significantly different from Fig. 4.1b where the bandwidth $h = 3.7$. The three eigenvectors used by KECA are plotted in red Fig. 4.5b. While they look very similar to the ones used by KECA in Fig. 4.2, they now correspond to the three largest eigenvalues, which means that uncentered kernel PCA will give the same clustering result as KECA.

Chapter 5

Novel Mean Shift Spectral Clustering

5.1 Introduction

Spectral methods is considered to be the state of the art when it comes to clustering, and they have been shown to produce good results in a wide variety of situations [28]. A drawback is that such methods require the eigendecomposition of the data affinity matrix, such as the kernel matrix for Kernel Entropy Component Analysis (KECA) and Kernel Principal Component Analysis (KPCA), and the Laplacian matrix for Laplacian eigenmaps. This means that spectral methods are not viable for large datasets.

In [28], Ozertem et al. presented a clustering method that enables the use of spectral methods for large datasets. The idea is to use a two-stage clustering approach. First the mean shift algorithm was used to provide an initial clustering. These clusters were referred to as *partitions* of the dataset. Then an affinity matrix is constructed based on these partitions rather than the full dataset. Spectral clustering is then performed based on this matrix. This enables the use of spectral methods for large datasets, provided of course that the number of partitions is sufficiently small. Ozertem suggested basing the spectral clustering on the symmetric normalized Laplacian matrix in Eq. (2.114). The method was called *Mean Shift Spectral Clustering* (MSSC).

This thesis is based on the idea of MSSC introduced in [28]. However, the focus and method described here differs from the original article in some aspects. In particular, KECA presented in Chapter 4 is introduced for the first time into the framework of Mean Shift Spectral Clustering. This chapter provides a brief summary of the work presented in [28] and how MSSC is implemented for this thesis.

5.2 A Two-Stage Clustering Approach

The clustering methods presented in Chapter 2 each have advantages and disadvantages. The k-means algorithm is fast and intuitive, but it might converge to a local optimum as seen in Fig. 2.3. Expectation Maximization (EM) presented in Section 2.3 should capture the linear structure of the data better as it uses Gaussian mixture models, but is slower and might yield poor results if the data deviates much from Gaussian distribution. Mean shift has the distinct advantage that it does not require the number of clusters as an input and is based on a non-parametric estimation of the data density function, thus it is a nonlinear local clustering method. This makes the mean shift relatively slow with a computationally complexity of $O(n)$ per sample for each iteration, where n is the number of data points. The number of output clusters depends on the number of iterations, the kernel used for density estimation and the bandwidth, also called the kernel size, parameter.

The spectral clustering methods have the distinct advantage over these algorithms in that they are easily able to handle non-linear cluster structures. As mentioned a major drawback is that they require the eigendecomposition of a $n \times n$ affinity matrix, which makes them unsuitable when the number of data points n is large. Finding the eigenvectors have computational complexity of $O(n^2)$ per vector. The upper limit of n is determined by the system used for the computations. In general it might help if the affinity matrix has some particular structure, particularly if it is sparse. Nevertheless, spectral clustering is best suited for smaller datasets.

In [28], a clustering method based on merging the clusters found by the mean shift algorithm was suggested. These m clusters, or partitions, are formed by the feature vectors that converge to the same mode. If one finds a representative partitioning of the dataset where $m < n$, these partitions can be used to form an $m \times m$ affinity matrix for spectral clustering. This gives a significant reduction in computational cost as the cost of calculating an eigenvector for the partition affinity matrix would be $O(m^2)$ rather than $O(n^2)$ for an affinity matrix based on the complete dataset. The spectral clustering stage merges statistically insignificant partitions into significant larger and more balanced clusters, which gives the clustering solution for the complete dataset.

5.3 The First Stage

It was noted in the article that it does not matter how the initial clustering assignment is obtained. Other clustering procedures can be selected for

partitioning the dataset, for instance due to computational complexity limitations or special heuristic rules for the task at hand. Hence the first stage could be performed by the k-means or EM algorithm presented in Chapter 2. K-means would do well if the datasets can be accurately described by a set of linearly separable spherical partitions, while the EM algorithm described in Section 2.3 would give partitions in terms of a mixture of Gaussians which would allow it to describe more advanced data structures.

The mean shift algorithm has some attractive properties that makes it particularly well suited for partitioning. As seen in Section 2.4, the algorithm is based on finding the modes in the probability distribution of the data. The overall data distribution is estimated by kernel density estimation, which is a non-parametric approach. Hence, no assumptions are made about the dataset. This is a big advantage compared to k-means and the EM algorithm as it can find partitions which are very different in shape and size. In addition one does not need to specify the number of partitions m in advance.

The number of partitions m should be so large that it accurately describes the structure of the dataset, yet small enough that it is possible to find the eigenvectors of the $m \times m$ affinity matrix. When using the mean shift algorithm, the number of partitions will depend on the kernel selected and the bandwidth used. Thus its performance depends heavily these choices. This is also the case for spectral methods, in both cases which kernel function to use is the most important parameter choice. There is no single unifying theoretical criterion for choosing the kernel in the present literature. Often Mercer kernels satisfying the conditions in Theorem 1, such as the widely used Gaussian kernel, are chosen [28].

When the kernel function is selected, one must choose the kernel size. There is a wide literature on how to select the kernel size and several alternatives were presented in [28], including variable sized kernels based on estimating the local covariance for the mode-finding vectors. This implies using a full bandwidth matrix as seen in Eq. (2.51). However, there is a clear trade-off between clustering performance and computational cost between variable and fixed kernel size. When variable sized kernels are used in the article, the local mean or covariance for each mode-finding vector is estimated based on some number of its nearest neighbours. For fixes size kernels, Silverman's rule of thumb was used:

$$h^2 = \frac{1}{n}\mathrm{tr}\left(\mathbf{\Sigma}_X\right)\left(4/((2n+1)n)\right)^{2/(n+4)} \tag{5.1}$$

where $\mathrm{tr}\left(\mathbf{\Sigma}_X\right)$ is the trace of the covariance matrix for the dataset. Silverman's rule is a suitable choice for unimodal data distribution, but might have problems with accurately describing densities with a more complex form [28].

In all experiments the Gaussian kernel was used.

5.4 The Second Stage

In standard spectral clustering algorithms the affinity matrix is constructed by evaluating pairwise similarities between samples [28]. For the kernel matrix all pairwise similarities are used, while the affinity matrix used for Laplacian eigenmaps depends on how the neighbourhoods are defined when constructing the adjacency matrix. For MSSC the affinity matrix is formed based on partitions which usually consisting of more than one data point. The affinity matrix could easily be constructed by using the modes found by the mean shift algorithm as cluster representatives. However, the article suggested using a metric based on the angle between the means of two partitions.

This metric is in fact the Cauchy-Schwarz divergence between densities, or rather its information potential V_{CS}, as defined in Section 4.4. If the partitions C_i and C_j have probability density functions (pdfs) $f_i(\mathbf{x})$ and $f_j(\mathbf{x})$, Eq. (4.14) can be written as

$$V_{\mathrm{CS}}(f_i, f_j) = \frac{\int f_i(\mathbf{x}) f_j(\mathbf{x}) d\mathbf{x}}{\sqrt{\int f_i(\mathbf{x})^2 d\mathbf{x} \int f_j(\mathbf{x})^2 d\mathbf{x}}} \tag{5.2}$$

Since we do not know the pdfs, they must be estimated from the dataset $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$. The natural choice is kernel density estimation. Then element i, j of the affinity matrix $\mathbf{A}$ is given by $A_{ij} = \hat{V}_{\mathrm{CS}}(f_i, f_j) = V_{\mathrm{CS}}(\hat{f}_i, \hat{f}_j)$ so

$$A_{ij} = \frac{\sum_{\mathbf{x}_k \in C_i} \sum_{\mathbf{x}_l \in C_j} K(\mathbf{x}_k, \mathbf{x}_l)}{\left(\sum_{\mathbf{x}_k \in C_i} \sum_{\mathbf{x}_{k^*} \in C_i} K(\mathbf{x}_k, \mathbf{x}_{k^*}) \right)^{1/2} \left(\sum_{\mathbf{x}_l \in C_j} \sum_{\mathbf{x}_{l^*} \in C_j} K(\mathbf{x}_l, \mathbf{x}_{l^*}) \right)^{1/2}} \tag{5.3}$$

where $K(\cdot, \cdot)$ is a kernel function suitable for kernel density estimation as defined in Section 2.4.1. It is possible to express Eq. (5.3) in terms of the elements of kernel matrix $\mathbf{K}$ of the dataset associated with the kernel function $K(\cdot, \cdot)$:

$$A_{ij} = \frac{\sum_{\mathbf{x}_k \in C_i} \sum_{\mathbf{x}_l \in C_j} K_{k,l}}{\left(\sum_{\mathbf{x}_k \in C_i} \sum_{\mathbf{x}_{k^*} \in C_i} K_{k,k^*} \right)^{1/2} \left(\sum_{\mathbf{x}_l \in C_j} \sum_{\mathbf{x}_{l^*} \in C_j} K_{l,l^*}) \right)^{1/2}} \tag{5.4}$$

The article noted that the kernel matrix $\mathbf{K}$ corresponding to the kernel used in the mean shift algorithm was calculated in the first iteration, as can be

seen from for example Eq. (2.58). Hence it is possible to express Eq. (5.4) in terms of $\mathbf{K}$ found by mean shift.

In [28] the final affinity matrix used was the normalized symmetric Laplacian matrix $\mathbf{L}_{\text{sym}}$ defined in Eq. (2.114). Written in terms of $\mathbf{A}$ found by Eq. (5.4) gives

$$\mathbf{L}_{\text{sym}} = \mathbf{D}^{-1/2}\,\mathbf{L}\,\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\,\mathbf{A}\,\mathbf{D}^{-1/2} \tag{5.5}$$

where $\mathbf{L}$ is the unnormalized graph Laplacian, $\mathbf{D}$ is the degree matrix and $\mathbf{A}$ is used as the weight matrix $\mathbf{W}$ corresponding to a *fully connected* graph between partitions according to the definitions in Section 2.6.2. All matrices are $m \times m$.

The article notes that any standard spectral clustering method can be used on the partition affinity matrix. However, while the MSSC method described in [28] is based on the normalized Laplacian matrix, the final result is not obtained using Laplacian eigenmaps as described in Section 2.6 or any other spectral methods. Rather, an algorithm based on finding the connected components of $\mathbf{L}_{\text{sym}}$ with complexity $O(m^4)$ was proposed. While it would make the algorithm impractical for large affinity matrices, it was reported to give good results when the matrices were small. Their experiments were based on this mode merging.

5.5 Proposed Changes in MSSC

The mode merging algorithm presented in the article will not be used in this thesis. This thesis will for the first time investigate truly two-stage spectral clustering, in the sense that the second clustering step will be conducted based on the *eigenvectors* of an affinity matrix. We propose to base the spectral clustering on KECA and KPCA with the partition kernel matrix based on the Cauchy-Schwarz divergence as defined in Eq. (5.4). From this matrix, a transformation of the partitions is found and then clustered together by using the k-means algorithm as suggested in [27] and [18].

Another important difference is how the challenge of finding a bandwidth that accurately describes the dataset is handled. In [28] three different suggestions are presented, one is using a fixed size kernel with a scalar bandwidth parameter h found by using Silverman's rule of thumb in Eq. (5.1). The two other suggestions were variable sized kernels where the kernel size for each mode finding vector based on estimating the mean or covariance of its k nearest neighbours. This was multiplied by a global scaling factor optimized using maximum likelihood. Their experiments compared fixed and variable sized kernels. For both cases the same kernel size was used for the mean shift algorithm and for constructing the affinity matrix.

In this thesis a fixed size kernel is used. The bandwidth matrix is chosen to be proportional to the identity matrix as defined in Eq. (2.52) so the bandwidth parameter is a scalar, h. However, here different bandwidths are used for each stage of MSSC. The motivation for this is that the mean shift algorithm should find a partition of the dataset and hence the kernel size should be small to accurately capture the *local* structure of the data. In the spectral stage these partitions should be merged, and to do this the kernel sized used should then describe the dataset on a *global* scale.

Several of the results presented in this thesis seeks to characterize how the performance of MSSC depends on these bandwidth parameters.

5.6 Parameters in Two-Stage Clustering

Using a combination of two clustering procedures will obviously increase the number of parameters one has to select. This makes the process of trying different parameters much harder as the number of possible combinations is quite large. In Chapter 2, the parameters for each of the clustering procedures are somewhat idealized. The focus is usually on one or two key parameters or options. For instance, for the k-means algorithm the key parameter is the number of clusters k, for mean shift the focus is on the kernel function, the bandwidth parameter and the option to use blurring. In Laplacian Eigenmaps, the neighbourhood definition and the edge weights are key properties (Section 2.6.2).

All of these clustering procedures have other parameters and options that could be or even needs to be specified. In Section 2.2, it was shown that the k-means algorithm can be derived from cost function optimization when the cost function is the sum of the squared Euclidean distance. However, the algorithm works for other distance measures and the intuition of assigning a point to the cluster that is closest (in that distance) is preserved. As discussed in Section 4.4, the cosine distance measure seems a better choice for the the Kernel Entropy Component Analysis (KECA) clustering method.

Also, as mentioned the k-means algorithm might converge to local minima of the cost function in Eq. (2.8) depending on how the cluster centers were initialized. One way of making the algorithm more robust is running the algorithm multiple times and selecting the best result. This might give an unrealistic picture of what we can expect of k-means when characterizing the algorithm and therefore examples of local minima convergence was shown and discussed, for instance in Fig. 2.3. However, when characterizing the two-stage clustering approach one should try to eliminate this weakness as it might randomly cause a worse result. This will make it harder to compare

parameter choices as well as comparing the overall clustering procedure to other methods.

For illustrative purposes, if we know the real labels of the dataset, one can run the algorithm several times and choose the best result. For example, if 25% of the possible combinations for selecting intial points leads to suboptimal results, independently selecting a new set of intial points and rerunning the algorithm five times and using the best result will make the probability of a suboptimal result less than 0.1%. In this thesis, when k-means is used as part of spectral clustering in the second step and the real data labels are known, the algorithm is run several times with different initializations and the best result is used. When the labels are not known, it is possible to use the cost expressed in Eq. (2.3) to choose the best result.

While the mode-finding vectors in the mean shift algorithm will converge to the same point for vectors in the same cluster for infinitely many iterations, one obviously needs to limit the number of iterations in some way. This leads to more parameters and options. The mean-shift code developed in this thesis for characterizing the Mean Shift Spectral Clustering approach has a user selected number of iterations and combines this with "adaptive thresholding" which seeks to group points together even when the mode-finding vectors have not fully converged.

For Laplacian eigenmaps, it was mentioned in Section 2.6 that there are three different versions of the Laplacian matrix one can use. When also considering the different ways of defining neighbours and edge weights seen in Section 2.6.1, there are a lot of parameter combinations to be tried, each of which can significantly influence the result. Therefore the spectral methods focused on in this thesis will be Kernel Principal Component Analysis (KPCA) and KECA. Here primary parameter choice is which kernel to use. For the two-stage approach, KPCA always uses the centered kernel matrix.

These examples are meant to illustrate the multitude of options a two-stage clustering presents. While much work has been done in this thesis to make these easily changeable in the code developed, the results will be based on varying only a few of these. We assume that the number of output clusters is known as this helps better characterize the method. Another common convention is to also set the number eigenvectors to use in KPCA or KECA, which gives the number of dimensions of the transformed dataset, equal to the number of input classes which must then be assumed to be known in advance. The reason for this is discussed in Section 4.4.

Chapter 6

Results

6.1 Examples to Illustrate MSSC

This section shows the clustering results using Novel Mean Shift Spectral Clustering (MSSC) used on three datasets, one well known pattern recognition benchmark in Section 6.1.1, one black and white image in Section 6.1.2 and a toy data example in Section 6.1.3. These datasets will be used to illustrate and discuss some important aspects of MSSC.

6.1.1 The Iris Dataset

The Iris dataset was first used by Fisher in [10]. The dataset has become a well known benchmark for machine learning and it can be found in the UCI repository [11]. The dataset consists of 150 measurements of sepal and petal width and length of three different types of the Iris plant (Iris setosa, Iris virginica and Iris versicolor). One of the classes is linearly separable from the other two [11]. This dataset will be used to illustrate some of the aspects of two-stage clustering.

Figure 6.1 shows the resulting number of clusters found by the mean shift algorithm in the Iris dataset, plotted as a function of the bandwidth (kernel size) parameter h. The kernel used was the Gaussian, which we know from Eq. (2.93) can be written as $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{1}{2}\left|\left|\frac{\mathbf{x}_i - \mathbf{x}_j}{h}\right|\right|^2\right)$. The number of mean shift iterations was kept constant at 100.

The blue line in Fig. 6.1 shows the total number of clusters while the dashed red line shows how many of these are single point clusters (outliers). For the smallest bandwidth $h = 0.01$ only a few points have been clustered together by the mean shift algorithm. Then as the bandwidth increases the number of clusters decreases and for $h = 0.35$ the dataset is reduced to two

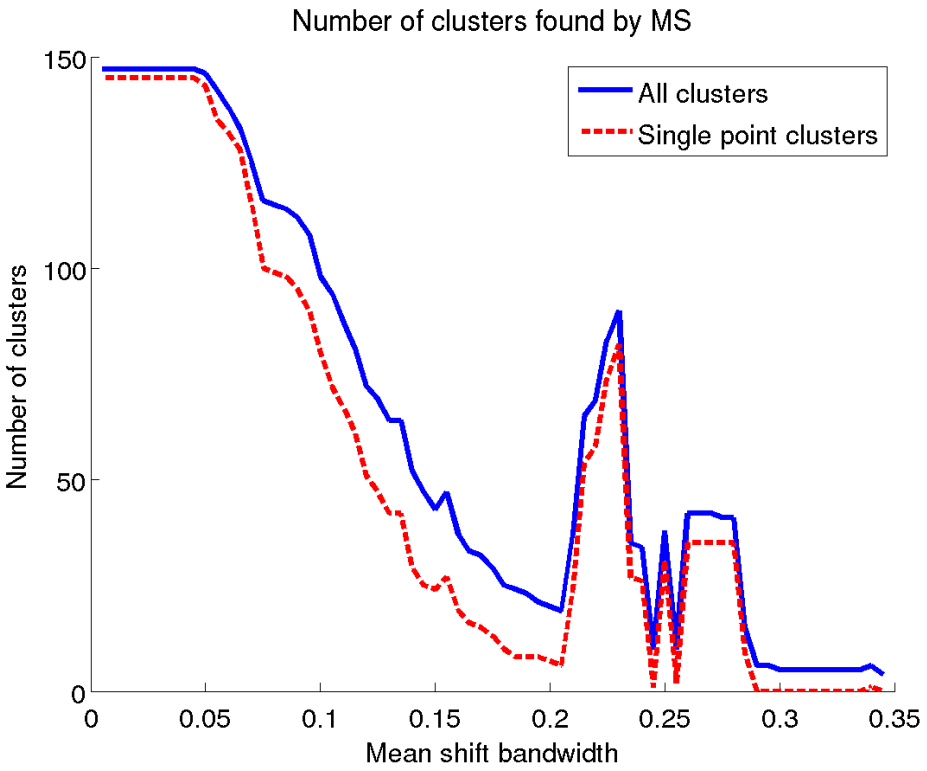


Figure 6.1: Number of clusters for different mean shift bandwidths.

clusters (not shown in the plot). We see that when the bandwidth is between $h = 0.10$ and $h = 0.20$ the curves showing the number of single point clusters and the total number of clusters seem to have the same shape. This indicates that the reduction of the total number of clusters is caused by outliers being merged with existing clusters.

The sudden variation in the number of clusters between $h = 0.20$ and $h = 0.30$ has to do with the number of mean shift iterations. When the kernel size increases modes are merged together, but the convergence is slow due to the mode finding vectors having to pass through regions with high density. This will be discussed in more detail later.

To try to predict such effects one would have to study the geometry of the dataset. A sudden increase in the number of clusters can be an indication that the kernel size is so large that the mode finding vectors are significantly influenced by the global structure of the data. When using mean shift in a two-stage clustering approach one wants the mean shift to characterize the local structure of the data, before using a spectral method to find the final clusters. In this setting, the number of clusters found by mean shift should

be small enough to efficiently use spectral methods, but not so small that it approaches the final number of clusters. The clusters found in the first step should capture the local structure of the data. A sudden increase in the cluster number as seen in figure Fig. 6.1 might be an indication that the bandwidth is becoming so large that the mode-finding vectors are significantly influenced by data points outside the local cluster.

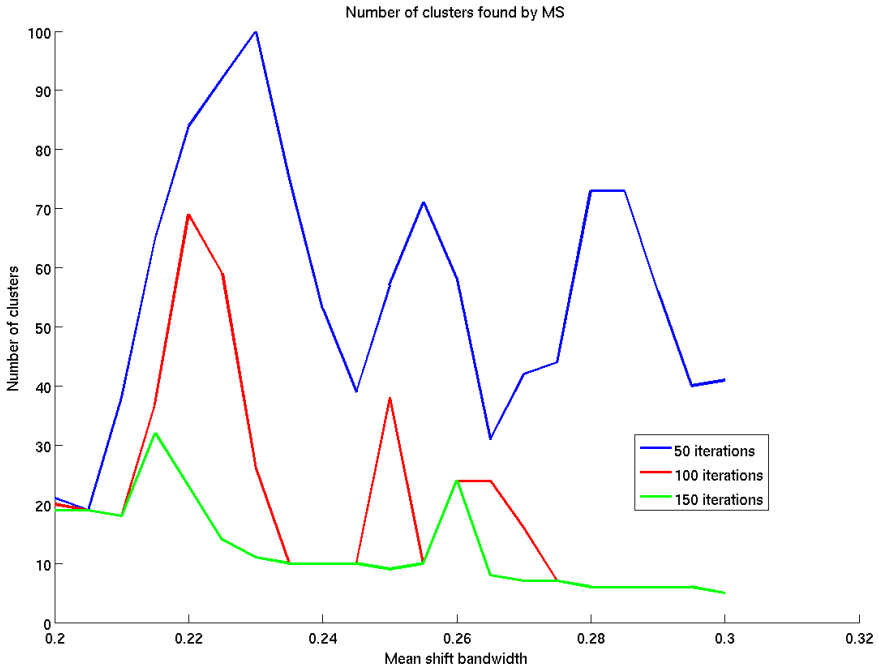


Figure 6.2: Number of clusters found by mean shift, different number of iterations.

In Fig. 6.2 the total number of clusters is plotted as a function of bandwidth for different number of mean shift iterations. We see that for 50 iterations, the number of clusters vary more than for 100 iterations. When the number of mean shift iterations is set to 150, the variations are less apparent. The cluster number is also influenced by how we define which mode-fining vectors have converged to the same mode. Looking for an increase in the cluster number to say something about what bandwidth to use is therefore a somewhat unreliable heuristic.

Figure 6.3 shows the accuracy in percent of correctly clustered points for two-stage clustering as a function of the bandwidth which is equal for mean shift and the spectral clustering. In the leftmost plot, the solid blue line is

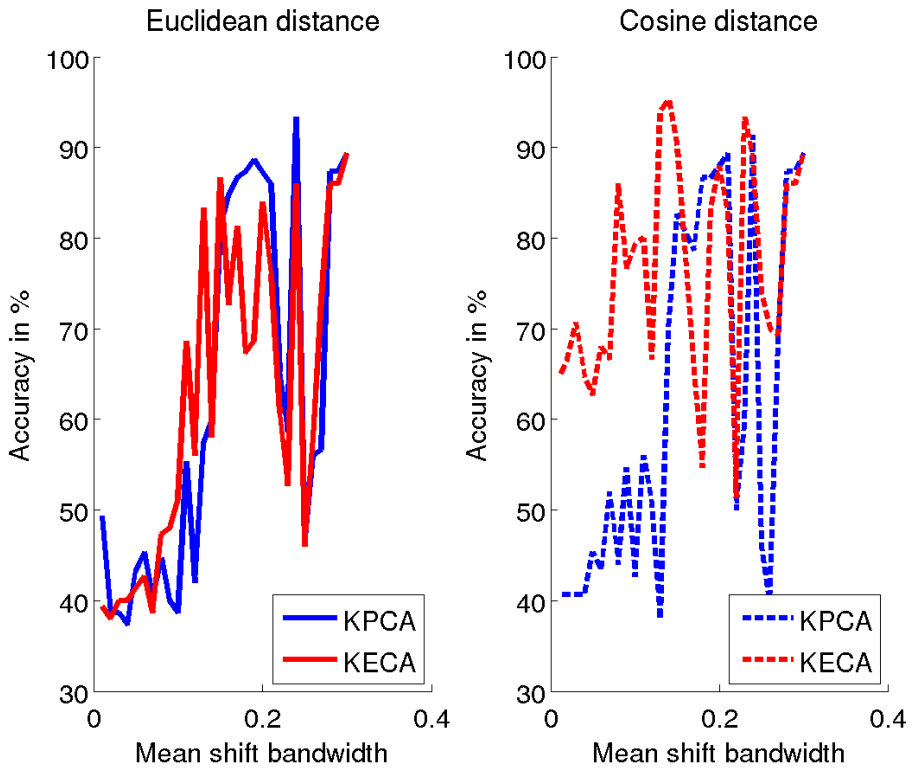


Figure 6.3: Classification accuracy as a function of mean shift bandwidth.

the accuracy using KPCA spectral clustering with the standard Euclidean distance for the k-means algorithm used for finding the final clusters, the solid red line is the accuracy of using KECA for the spectral step with the same distance measure. For the rightmost plot, the distance used in the spectral clustering is the cosine distance as defined in Eq. (4.16), the dashed blue line is the accuracy for KPCA and the dashed red line is for KECA. While Section 4.4 suggests using the cosine distance for KECA, we will look at which distance measure gives the best result for this dataset, both for KECA and KPCA.

As we can see in Fig. 6.3, the cosine distance measure overall gives slightly better results than the Euclidean distance for KECA. However, the classification accuracy varies too much to draw conclusions regarding which of the spectral clustering methods perform best for this dataset.

The kernel matrix used in the second step is based on calculating the Cauchy-Schwartz (CS) divergence between the clusters found in the mean shift step according to Eq. (5.3). For the result in Fig. 6.3, the kernel used for defining the kernel matrix is the same as for mean shift. As noted in

[28], the full kernel matrix is calculated in the first step of the mean shift algorithm using the Parzen windowing kernel. It is therefore possible to reuse these calculations as seen in Eq. (5.4), which may reduce the running time of the algorithm. This of course means that the same kernel size will be used for both steps.

While this often is more computationally efficient, it might be problematic as one wants the kernel size used for the mean shift step to find the local modes in the dataset, while the kernel matrix should characterize the "global" structure of the data. Hence it would be reasonable that the bandwidth used for the second step is bigger than for the first step since we want the spectral step to merge the mode partitions (clusters) found by mean shift in the first step.

The code developed for the thesis seeks to utilize the calculations done in the mean shift step by noticing that the Gaussian kernel function can be expressed as a function of the Euclidean distance between the points and the bandwidth parameter when using a diagonal bandwidth matrix as seen in Eq. (2.93). In the code, the first mean shift iteration builds and saves the the distance matrix. Then one can easily reduce these distances with a different kernel size when calculating the CS divergence.

Regardless if one chooses to store the kernel matrix and use the same bandwidth for both steps, this requires storing a $n \times n$ distance or kernel matrix. This means the dataset should not be too large. For instance, for storing the full kernel matrix for a dataset of $n = 50000$ data points with double precision (each element is 8 bytes), requires 50000^2 data points $*$ 8 bytes/data point = 20GB[1]. While it is possible to use a smaller precision (in single precision each number is 4 bytes), one might ask if it is worth storing a large distance matrix to save computational time.

In Fig. 6.4 the percentage of correctly classified points is plotted as a function of increasing the bandwidth for the spectral step when the mean shift bandwidth is kept constant at $h_{\text{ms}} = 0.12$. The spectral bandwidth varies between 0.1 and 10 in steps of 0.1. Like Fig. 6.3, the left subplot corresponds to using the Euclidean distance while the right subplot is for the cosine distance.

We see that in both cases the clustering accuracy drastically improves once the spectral bandwidth becomes large enough. For KECA this is around $h_{\text{KECA}} = 1.2$ for both distance measures. The accuracy of KPCA improves much sooner, at around $h_{\text{KPCA}} = 0.4$. KPCA has a slightly higher, and more stable, overall accuracy when using the Euclidean distance than the cosine

[1]This could be reduced by using the fact that the distance matrix is symmetric and hence only requires storing $\frac{n(n+1)}{2}$ unique elements, but the point remains the same.

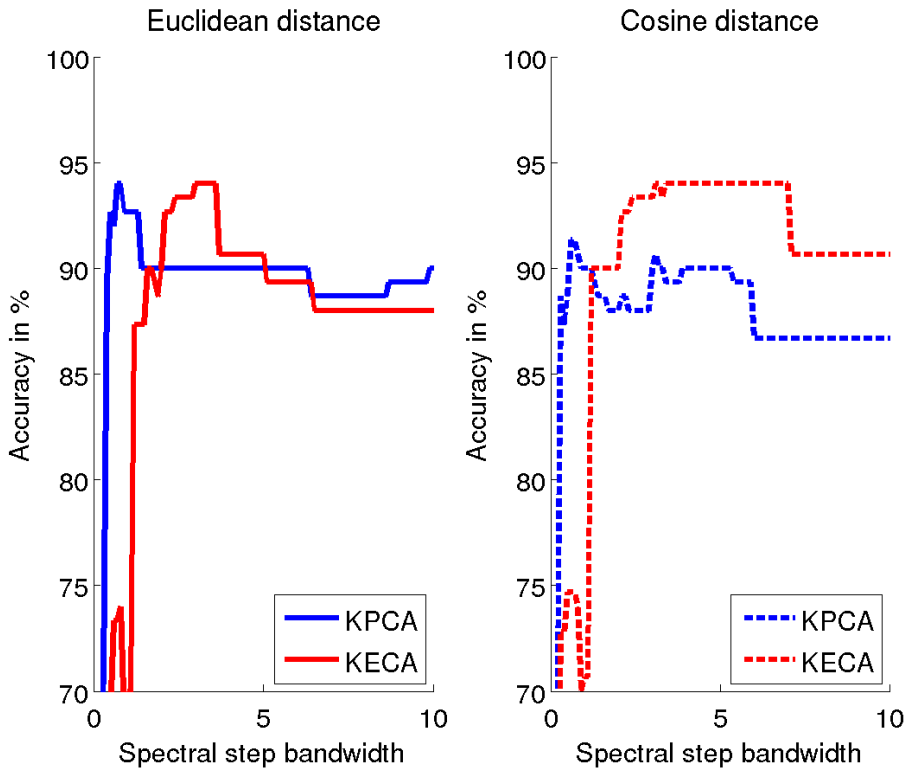


Figure 6.4: Classification accuracy as a function of spectral bandwidth.

distance in k-means. The opposite is true for KECA. In this case, when the kernel size is larger than 2.0, the KECA method with angular distance gives the best result. The k-means algorithm has an accuracy of 89.33% for this dataset.

Figure 6.5 shows an image plot where the colour indicates the percentage of correctly clustered points. The mean shift bandwidth varies between 0.01 and 0.30 in steps of 0.01, increasing from top to bottom along the vertical axis of the plots in Fig. 6.5. The horizontal axis represent spectral bandwidth between 1.0 and 5.0, increasing from left to right in steps of 0.2. Both spectral methods used the same partition found by the mean shift algorithm in 100 iterations and the Euclidean distance is used in k-means to obtain the final clustering result.

The result obtained using KPCA seems quite stable at around 90% accuracy, except when the mean shift bandwidth $h_{\text{ms}} \geq 0.20$. We find the reason for these variations in Fig. 6.1; the number of clusters found by the mean shift algorithm suddenly starts to oscillate at this bandwidth. The best accuracy is obtained in this region with a maximum of 98.0% accuracy. As

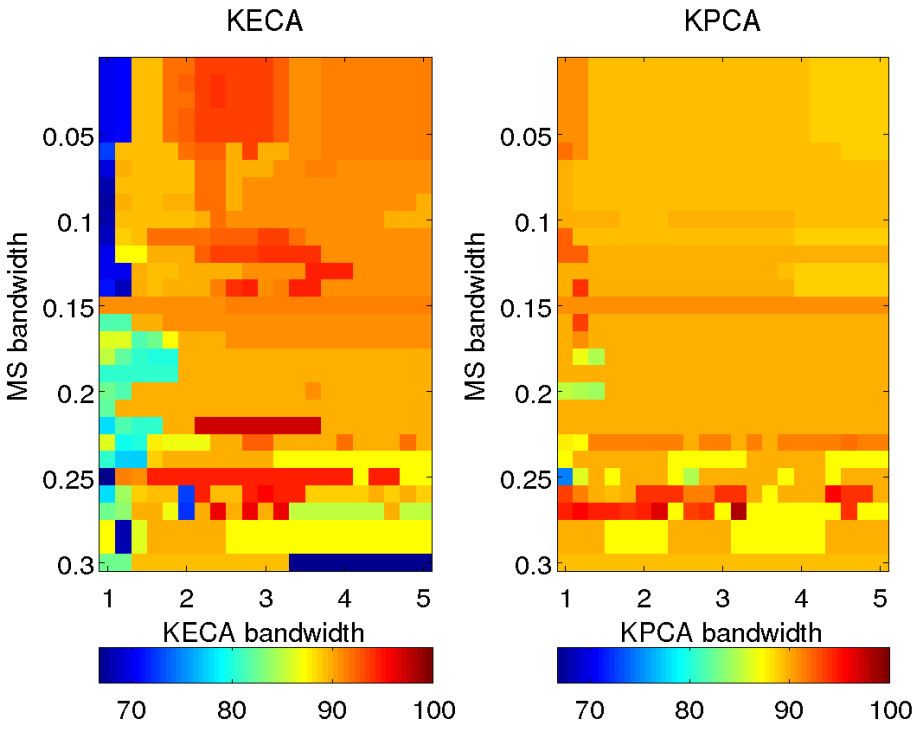


Figure 6.5: KECA and KPCA classification, Euclidean distance used for spectral step.

seen in Fig. 6.2 it is possible to negate this effect by increasing the number of iterations. The effect of the varying number of clusters for $0.20 \leq h_{\text{ms}} \leq 0.30$ is also apparent in the KECA accuracy.

Figure 6.6 is the same as Fig. 6.5, except that an angular distance is used to obtain the final clustering result. We notice the apparent lack of structure in the KPCA clustering accuracy. Clearly the cosine distance measure is not a good choice for KPCA in this case. For KECA however, the result is better and more consistent and the highest accuracy achieved is 97.77%.

The best result for KECA is actually obtained when the mean shift bandwidth $h_{\text{ms}} = 0.22$ as seen from the deep red line in the lower half of the left subplot. When we look at Fig. 6.2, we see that this corresponds to approximately 70 partitions and that when the bandwidth increases to $h_{\text{ms}} = 0.24$ the number of clusters drops to just over 10 and the result becomes worse. The best region for KECA is for small mean shift and large spectral bandwidth as seen in the upper right region of the KECA subplot. This corresponds well with the intuition to use a small bandwidth to capture the local structure in

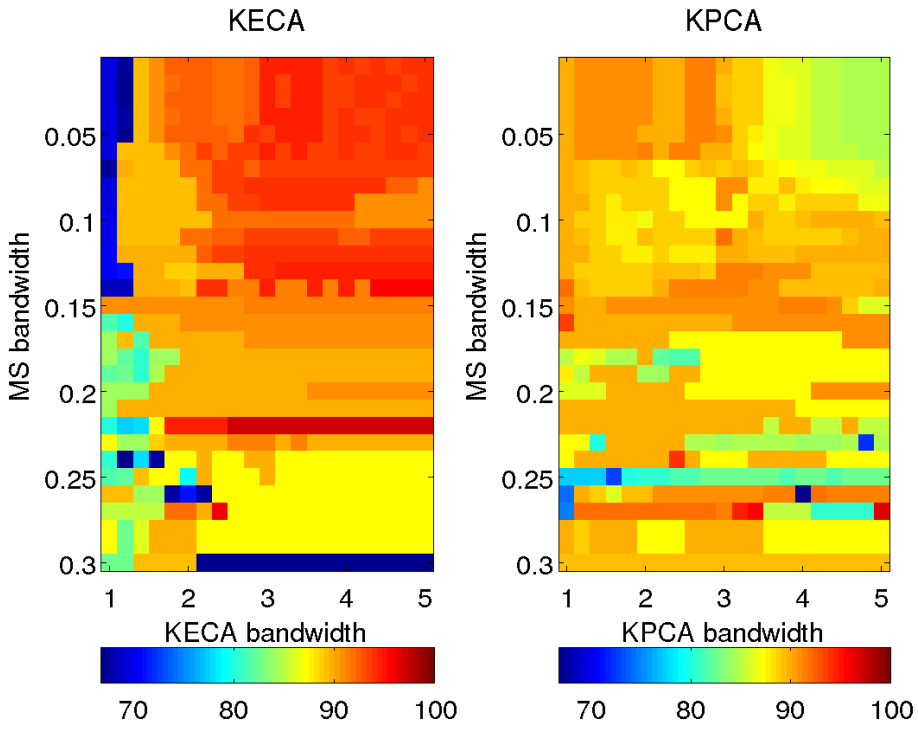


Figure 6.6: KECA and KPCA classification, cosine distance used for spectral step.

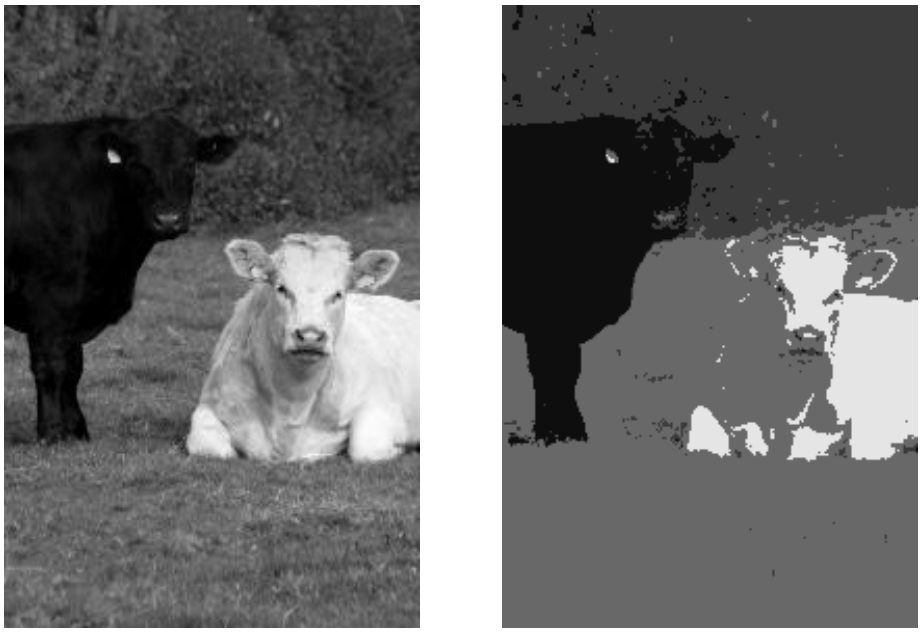
mean shift and then a large bandwidth to merge partitions in KECA.

For this dataset, it is clear that the cosine distance measure is best for KECA whereas the Euclidean distance is best for KPCA. KECA is able to obtain the highest accuracy, but is more sensitive to both kernel size parameters.

6.1.2 The Cows Image

One of the advantages of a two-stage clustering approach is the ability to use spectral methods on large datasets, such as images. This is demonstrated on a black and white image of two cows. The image should not be particularly difficult to cluster or segment with traditional methods. A simple thresholding would give good results. It is therefore well suited to illustrate some of the aspects of Mean Shift Spectral Clustering (MSSC) when it comes to clustering images.

Figure 6.7a shows the original image obtained from [1]. The image is 200×



(a) Original. (b) Four clusters.

Figure 6.7: The black and white cows picture.

300 pixels with 8-bit pixel intensities (256 different levels). In addition to the intensities, the coordinates of each pixel are included in the feature vectors. Thus the resulting dataset has $n = 60000$ feature vectors of dimension $l = 3$.

The different nature of the intensity and spatial features has to be compensated for by proper normalization [7]. A simple heuristic was used; each coordinate component was scaled to range from zero to some upper limit. An upper limit of 0.25 was used for the result seen in Fig. 6.7b. Hence feature vectors which have similar intensities, but corresponds to pixels located on opposite ends of the image, will be much closer than feature vectors corresponding to neighbouring pixels with a big difference in pixel intensities[2].

The mean shift bandwidth used for the result in (b) was $h_{\text{ms}} = 0.06$. Using 100 iterations, the mean shift algorithm outputs four partitions, which are represented by the average pixel intensity assigned to that cluster shown in (b). When the desired number of output clusters is set equal to four, the spectral clustering step will in this case do nothing and the final result will be completely determined by the mean shift algorithm. The partitions found by the mean shift algorithm sets an upper limit for the number of output clusters.

For a two-stage approach, it is therefore desirable to have the mean shift

[2]Pixel intensities are scaled to range from 0 to 1.

algorithm find a number of partitions greater than the desired number of output clusters. The number of partitions m should satisfy two criteria. First, it should be small enough to efficiently be able to find the eigenvectors for the $m \times m$ kernel matrix. Second, it should be so large that the partitions accurately describes the structure of the data. At first glance this may seem like conflicting demands, but together they actually specify an interval of acceptable partition numbers.

While the first criterion is determined by the processing system (computer and programs) used, and can therefore easily be found, the second criterion is harder to determine as it depends on the dataset. The only certainty is that it should be larger than the desired number of output clusters. Even if we do explore the data, it is not easy to establish the exact number of partitions needed to get an accurate representation.

The number of clusters found by mean shift will depend on the parameters used. While some parameters are easy to specify, kernel optimization is a tedious task and the lack of general and practical way of choosing kernels and the bandwidth parameter is a weakness of the mean shift algorithm [28]. For MSSC the spectral clustering step will merge partitions, and it is not problematic if the mean shift algorithm splits clusters into two or more partitions. Therefore we have more freedom in the choice of kernel size, as long as it provides a sufficient number of partitions. Hence the kernel size for the first stage should be smaller than one would use to cluster the dataset directly. In this thesis $m = 2500$ was used as an upper limit of mean shift clusters.

Figure 6.8 shows the number of partitions m as a function of the mean shift bandwidth h_{ms} varying between 0.01 and 0.10 in steps of 0.01. The green lines represent the number of clusters found by the regular mean shift algorithm, while the red are the results when using blur (described in Section 2.4.2). The stapled lines are the results with 50 mean shift iterations and the solid lines correspond to 100 iterations. Note that Fig. 6.8 has a logarithmic y-axis.

The regular mean shift algorithm with 100 iterations finds over 50000 partitions for $h_{\text{ms}} = 0.01$ and $h_{\text{ms}} = 0.02$. As the kernel size increases, the number of clusters decreases to just over 28000 for $h_{\text{ms}} = 0.03$ and $h_{\text{ms}} = 0.04$. When the kernel size becomes $h_{\text{ms}} = 0.05$, the mean shift algorithm returns $m = 4$ partitions. The result when using 50 iterations is similar. Figure 6.7b shows the four clusters found with 100 iterations for $h_{\text{ms}} = 0.06$.

While this extreme sensitivity to the bandwidth might be useful in evaluating whether the mean shift algorithm has converged to a stable number of clusters, it is problematic in our setting of a two-stage approach. Here the mean shift algorithm has limited the final clustering result to a maximum

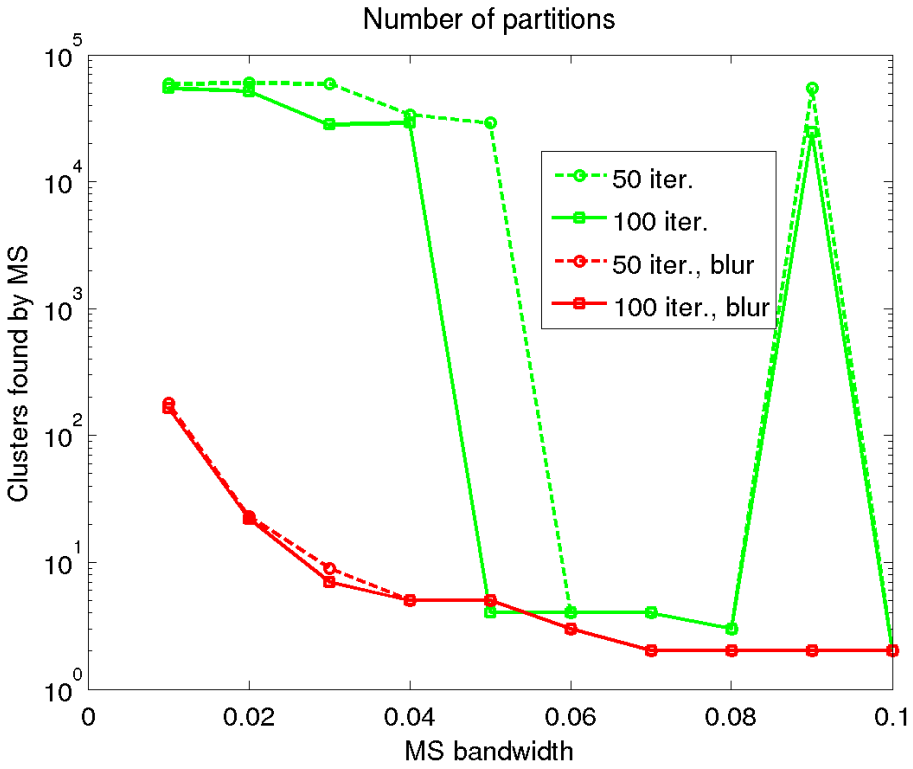


Figure 6.8: Number of clusters found by mean shift.

of four clusters. Even if the final result should consist of four clusters, one might be able to obtain a better result if the spectral step merged a larger number of partitions.

When the kernel size increases to $h_{\text{ms}} = 0.09$, the regular mean shift suddenly finds $m = 54988$ partitions with 50 iterations and $m = 24470$ for 100 iterations. This is the same phenomenon seen for Iris dataset described in Fig. 6.1. Here however, when the number of iterations is increased to 150, mean shift still finds over 24000 partitions. This is caused by the large kernel size merging what, for a density estimate using a smaller kernel size, were two different modes. Recall that the mode-finding vectors move very short distances close to a mode and far in low density regions. This can be seen from the denominator of Eq. (2.58). Thus the mode-finding vectors of the partitions merged need to move through the high density region that for smaller kernel sizes was a mode.

In this case, for $h_{\text{ms}} = 0.08$, 3 partitions were found with both 50 and 100 iterations. These consisted of 25757, 29210 and 5033 feature vectors. Increasing the kernel size causes two of these modes in the pdf estimate

obtained with $h_{\text{ms}} = 0.08$ to merge to a mode somewhere between them in the density estimate using $h_{\text{ms}} = 0.09$. Then points in the outskirts of these modes need to move through a high density region to "escape". This requires many iterations as the mode-finding vectors move very slow in such regions. Thus when the iterations stop, some of the mode-finding vectors will have converged to the joint mode, while others will be somewhere along the way.

While the distances between mode-finding vectors which have stopped in their tracks may be small, it will be several orders of magnitude larger than for those that have converged to a mode. The mean shift developed for this thesis uses the relative distance between vectors to cluster to avoid setting an arbitrary definition what is close for each dataset[3]. An example of the mode-finding vectors moving at different speeds in different density regions can be seen in Fig. 2.9.

Of the $m = 24470$ partitions found with 100 iterations, the largest contains 30519 of the pixel feature vectors, the second largest partition has 4985. The remaining partitions contain on average 1.0011 feature vectors. To avoid phenomenon, one has to change the kernel size or the number of iterations has to be increased significantly.

Another example of how important the number iterations is can be seen for $h_{\text{ms}} = 0.05$ where an additional 50 reduces the number of partitions by over 28000. However, as one mean shift iteration for this dataset takes just under 2.5 minutes on the system used[4], increasing the number of iterations by 50 means that the processing takes 2 hours longer to complete. Due the large number of iterations required and the extreme sensitivity to kernel size if this is not met, it is better to use blurred mean shift for this dataset.

The blurred mean shift plotted in red in Fig. 6.8 gives a much more stable number of output clusters, for $h_{\text{ms}} = 0.01$, 50 iterations with blur finds $m = 180$ partitions. This results in a 180×180 kernel matrix which poses no problems for the spectral clustering step. When the bandwidth increases to 0.02, the number of partitions is reduced to 23 before it continues to decrease to 9 and 5 for the next two bandwidths. Except for the smallest bandwidth where 50 iterations gives 17 more partitions than the result for 100 iterations, the biggest difference in number of partitions is 2.

[3]For example, while an Euclidean distance of 0.5 between two vectors can be considered close when the vector elements have values in the range of $\pm 10^6$, it should be considered a large distance in datasets where the each vector coordinate is between 0 and 1. Normalization helps, but does not fix the real challenge. There is no universal distance large enough to contain all clusters yet small enough to sperate between them common for all datasets, even normalized ones.

[4]IntelCore i7CPU 860 @ 2.80 GHz, 4.00 GB RAM, 64-bit Windows 7 SP1, MATLAB 2011a.

If the tendency seen in Fig. 6.8 applies for image datasets, one might consider using blurred mean shift with a relatively small number of iterations as a general strategy for the first stage.

Figure 6.9 shows the result of Mean Shift Spectral Clustering (MSSC) on the original image seen in Fig. 6.7a. A four cluster solution was selected manually in advance. For each pixel, the intensity is set equal to its cluster mean. In Fig. 6.9, (a) is the result of using mean shift with blur and 50 iterations. The Gaussian kernel is used with bandwidth $h_{\text{ms}} = 0.02$ which gives $m = 23$ partitions. These are shown in different colours in (b). Eight of the partitions contains less than 40 pixels, and of these five have less than 10 pixels. We notice that the mean shift algorithm has grouped most of the pixels in the black cow together, except for the ear tag, part of the foot and around the nose. Some pixels in the background are also included in this partition. The white cow is split into several partitions. Most of the background is divided into large coherent sections.

In (c) and (d) KECA is used in the spectral step, while (e) and (f) shows the result using KPCA. The spectral bandwidth is 0.2 for (c) and (e), and 1.0 for (d) and (f). For both spectral clustering methods, the k-means algorithm is run 500 times and the result with the lowest cost in terms of the distance function used (cosine for KECA and Euclidean for KPCA) is chosen for the final result.

The results in (d), (e) and (f) look quite similar. The partition of the black cows leg has been merged with the rest of the cow. The large background sections have been combined into the dark gray cluster, which also contains part of the shadowed region around the neck of the white cow, which we see from (b) is caused by this region being in the same partition as the grass between the cows. Other parts of the shadowed region of the cow is grouped in the light gray class along with a small speckled region of the grass. The rest of the white cow is clustered in the off-white cluster. This cluster also contains the black cows ear-tag.

The result when using a smaller kernel size for KECA shown in (c), merges the dark purple partition in (b) with the black cow partition. That the final results with KECA in (c) and (d) are different, while MSSC with KPCA in (e) and (f) are the same, is reminiscent to what we saw for the iris dataset i Fig. 6.5 and 6.6; KPCA appears more stable to variations in bandwidth than KECA.

(a) Mean shift result.

(b) The 23 partitions.

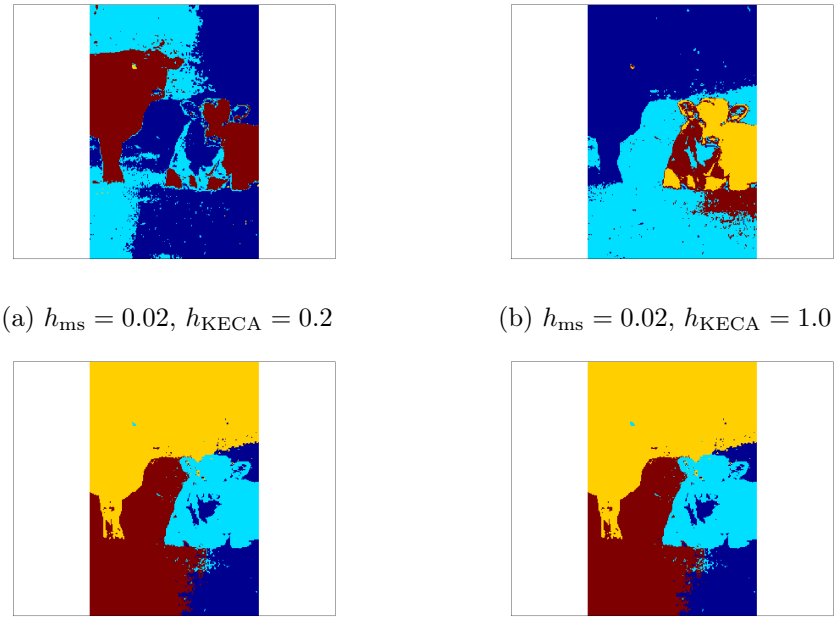
(c) $h_{\text{ms}} = 0.02$, $h_{\text{KECA}} = 0.2$

(d) $h_{\text{ms}} = 0.02$, $h_{\text{KECA}} = 1.0$

(e) $h_{\text{ms}} = 0.02$, $h_{\text{KPCA}} = 0.2$

(f) $h_{\text{ms}} = 0.02$, $h_{\text{KPCA}} = 1.0$

Figure 6.9: Four cluster results for KECA and KPCA.



(a) $h_{\text{ms}} = 0.02$, $h_{\text{KECA}} = 0.2$

(b) $h_{\text{ms}} = 0.02$, $h_{\text{KECA}} = 1.0$

(c) $h_{\text{ms}} = 0.02$, $h_{\text{KPCA}} = 0.2$

(d) $h_{\text{ms}} = 0.02$, $h_{\text{KPCA}} = 1.0$

Figure 6.10: Larger pixel coordinate range.

Figure 6.10 shows the clustering result when the pixel coordinates are scaled to be between 0 and 0.50, double the range used for the results in Fig. 6.7 and 6.9. The mean shift step divides the image into 78 partitions, three times more than with a coordinate range heuristic of 0.25. KECA is used in the spectral step for (a) and (b), and KPCA in (c) and (d).

The parameters used are the same as for Fig. 6.9, with Fig. 6.9c corresponding to Fig. 6.10a, Fig. 6.9d to Fig. 6.10b and so on. The four clusters are shown in different colours as partitions with different intensities are merged, which would give the images poor contrast as the cluster means would be similar, at least compared to the results in Fig. 6.9. In all cases, the result is worse than seen in Fig. 6.9. We also notice that again KPCA gives the same result, while KECA has different clustering result depending on the kernel size.

(a) Original. (b) K-means result. (c) MSSC result.

Figure 6.11: Comparison with k-means.

The original image, shown again in Fig. 6.11a, appears to have four distinct, coherent regions; the grass, the background and each of the two cows. Thus it should be possible to obtain a good result with a simpler clustering algorithm like k-means. Fig. 6.11b shows a $k = 4$ cluster solution when using the k-means algorithm directly on the dataset. It was run several times with different initializations and the result with the lowest k-means cost was chosen. For comparison, the result from Fig. 6.9f is shown in (c).

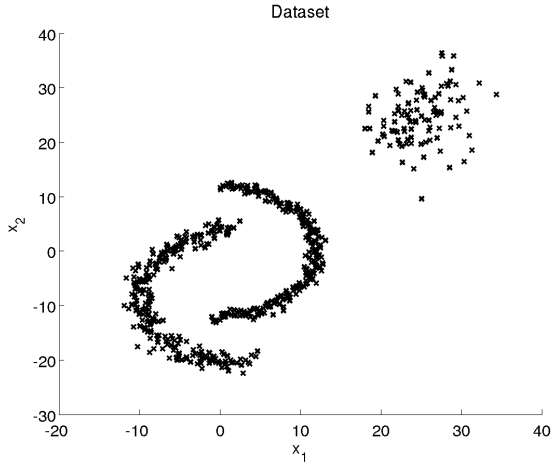
Which of the two results (b) and (c) is better would depend on what criterion is used. It is clear that visually the k-means clustering represents the original image better. The point that is important to note is that, even for this simple black and white image, MSSC gives a different clustering than k-means.

It should be noted that when increasing the coordinate scale range heuristic to 0.5, as done in Fig. 6.10, the k-means algorithm also gives a different clustering result than the results shown there. However, it is not the same as in Fig. 6.11b and the black cow is merged with parts of the background.

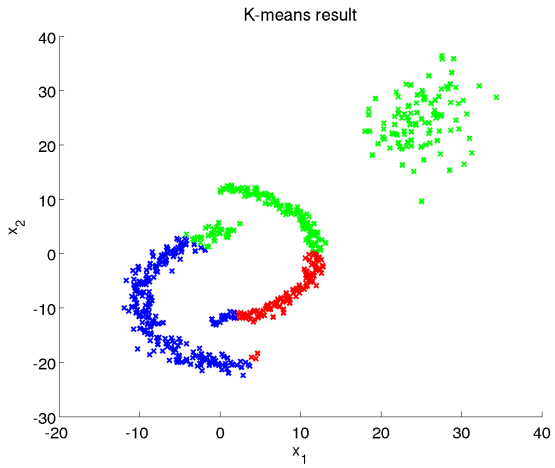
6.1.3 Toy Data Example

Figure 6.12 shows the toy data discussed as an example of clusters that are not linearly separable in Chapter 1. The dataset, also seen in Fig. 1.1b, is generated as two half circles each with 250 points and different added Gaussian noise, while the points in the upper right corner of (a) is 100 points form a third Gaussian distribution.

The result of running the k-means algorithm 1000 times and using the clustering with the lowest cost according to Eq. (2.8) is shown in (b). The different colours represent the $k = 3$ different clusters. This is not a good



(a) Toy data.

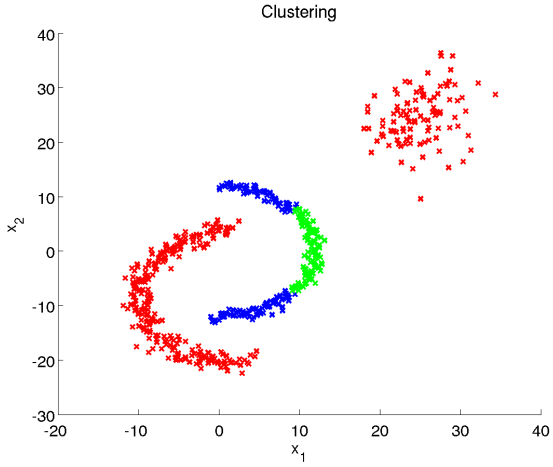


(b) K-means.

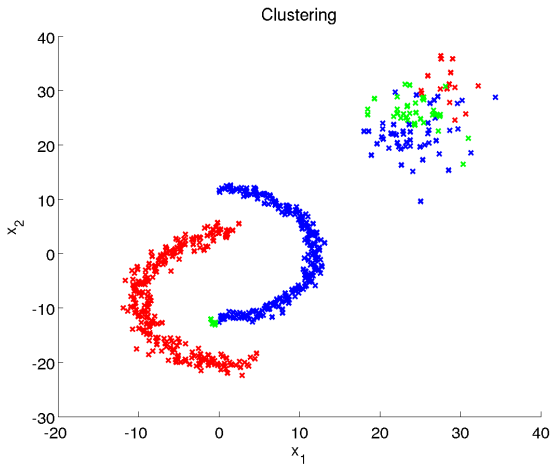
Figure 6.12: Plane.

clustering when we consider how the data was generated. When the plot was presented in Fig. 1.2, it was used as a motivation for using spectral clustering methods as they are able to handle non-linear cluster structures.

However, the spectral clustering results in Fig. 6.13 reveals that neither KPCA nor KECA were able to separate the two half circles and the noise perfectly. The kernel size was equal for both methods with $h = 2.0$. The KPCA result in (a) did cluster the two half circles in different clusters, but the right half was divided in three parts with the ends in one cluster and the middle section in another. The noise was clustered together with the right half circle. KECA is able to cluster the to half circles in separate groups, except for the tip of one which is in a third cluster. The noise however is



(a) KPCA.



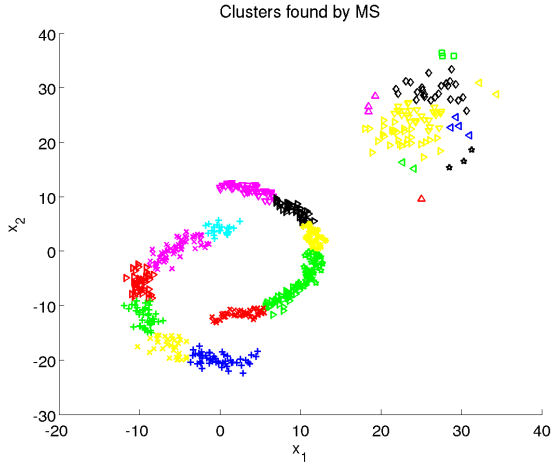
(b) KECA.

Figure 6.13: Spectral clustering with $h = 2.0$.

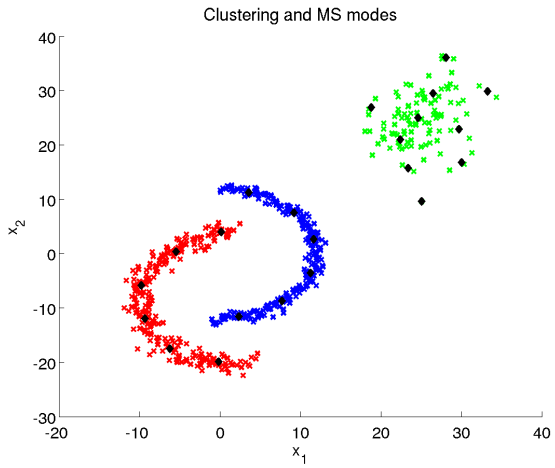
split between the three clusters.

Figure 6.14 shows the result of MSSC. Using 50 blurring iterations with kernel size $h_{\text{ms}} = 1.0$, the mean shift algorithm finds 22 partitions. These are plotted with different markers (note that some are very similar) in (a). Each of the two halves are represented by six partitions while the multivariate normal cluster has ten partitions. This is because the noise added to the half circles has lower variance, and combined with the higher number of points in these groups, the density estimate is relatively smooth in these regions.

The clustering result by using KECA on the kernel matrix found from Eq. (5.3) using these partitions is shown in (b). The kernel size used was the same as when using spectral clustering directly in Fig. 6.13b, $h_{\text{KECA}} = 2.0$.



(a) Partitions.



(b) MSSC.

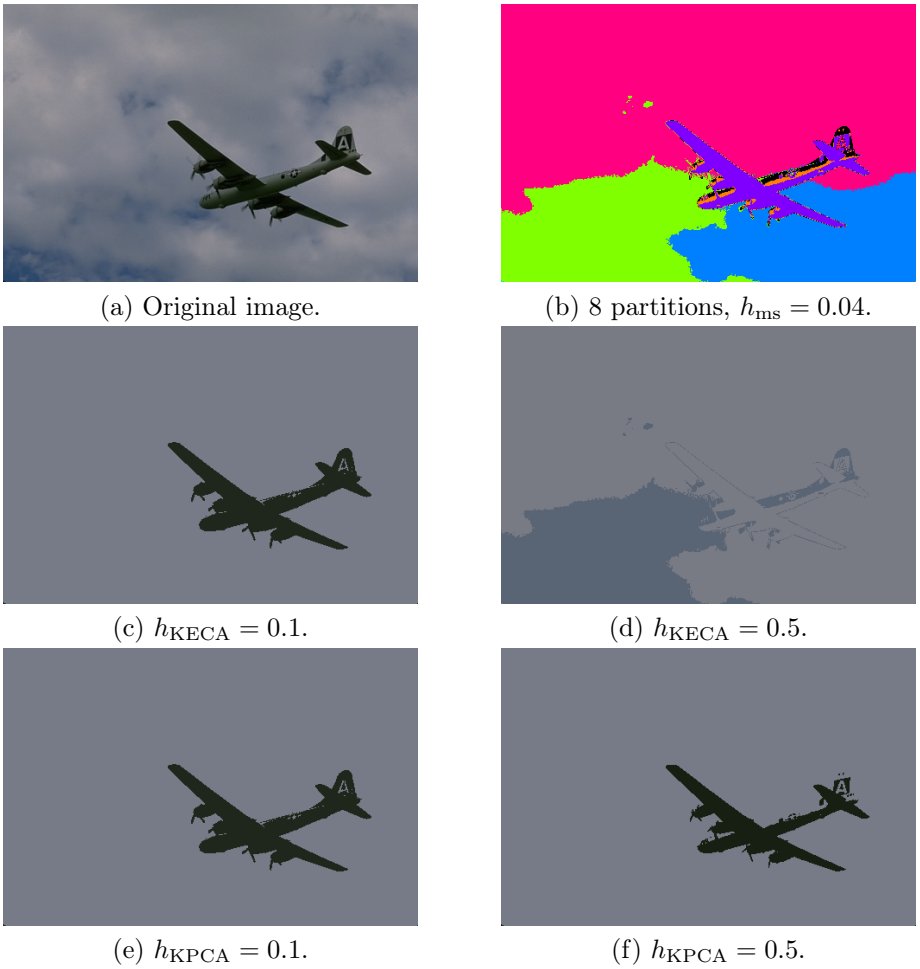
Figure 6.14: Clustering results.

In this case, MSSC gives the same result when KPCA (with the same kernel size) is used. For these parameters, MSSC performs better than using any of the two spectral clustering methods directly.

6.2 Image Segmentation

Section 6.2 contains image segmentation results on two colour images obtained from the Berkeley Segmentation Dataset and Benchmark [23]. Each image is used to discuss a particular aspect of MSSC. In all results, the clusters are represented by the average pixel intensity for that cluster.

6.2.1 Plane Picture



(a) Original image.

(b) 8 partitions, $h_{\text{ms}} = 0.04$.

(c) $h_{\text{KECA}} = 0.1$.

(d) $h_{\text{KECA}} = 0.5$.

(e) $h_{\text{KPCA}} = 0.1$.

(f) $h_{\text{KPCA}} = 0.5$.

Figure 6.15: Plane.

Figure 6.15 shows the MSSC segmentation result of the picture in (a) obtained from the Berkeley Segmentation Dataset and Benchmark [23]. Mean shift with bandwidth $h_{\text{ms}} = 0.04$ blur and 50 iterations is used to obtain eight partitions shown with different colours in (b). The image is 481×321 pixels, which means that the there are $n = 154401$ feature vectors in the dataset. In addition to the red, green and blue (rgb) intensity levels, which are between 0 and 1, each feature vector contains the x and y coordinate. The coordinates are scaled to be between 0 and 0.33, slightly larger than in Section 6.1.2 where this heuristic was set to 0.25.

KECA is used in the spectral step with $h_{\text{KECA}} = 0.1$ in (c) and $h_{\text{KECA}} = 0.5$ in (d). The segmentation is clearly better for the smaller kernel size

where the black, orange and purple partitions of (b) are merged together and the plane is clearly seperated from the other cluster corresponding to the background. In (d) however, the purple class is merged with the two background partitions blue and pink. Since both clusters now contain part of the background and part of the plane, the average pixel intensities are gray. We notice that it is possible to see the contours of the plane because the pixels along the edge of the wing belong to different partitions.

The MSSC result with KPCA and the same spectral bandwidths are shown in (e) and (f). With a spectral bandwidth of 0.1, KPCA merges the same partitions as KECA. The KPCA result is also worse when $h_{\text{KPCA}} = 0.5$, but the segmentation in (f) is better than in (d). Here the black and orange fuselage partitions are merged together, while the black is included with the background partitions. For all results the k-means algorithm in the spectral stage was run 500 times and the result with lowest cost function was chosen.

6.2.2 Water Buffalo Picture

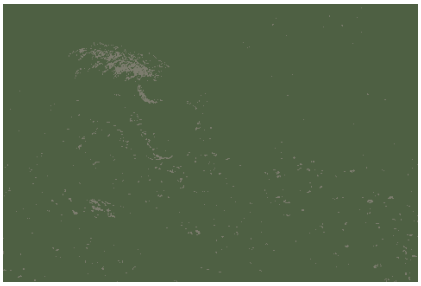
In Fig. 6.16 we study the how the two spectral clustering methods merges different partitions (b) of the original image (a). As in Section 6.2.1, the mean shift algorithm used blur and ran for 50 iterations and the coordinate scale heuristic was set to 0.33. With a kernel size of $h_{\text{ms}} = 0.02$, 77 partitions were found. These are represented by their average pixel intensity in (b). All results used a kernel size of $h = 0.1$ and 500 k-means initializations (5000 for the ten segments result) in the spectral stage.

The left column, (c), (e) and (g) are the results for two, four and ten segments found by using KPCA on the partitions in (b). The KECA segments in (f) and (h) appear to to give a better representation of the image than the corresponding KPCA results. The difference is best seen for four segments, where KECA has one of the buffalo partitions as its own segment, in addition to the general background and two gray segments which are used both for the water and the animal. The KPCA result in (e) has one large segment for the water and three segments with gray average pixel intensities.

Neither of the two cluster results in (c) and (d) are good representations, even though both have the top of the animal in a smaller cluster along with other bright parts of the image. The KPCA result gets better when the number of segments is increased to ten, but large parts of the animal is still merged with the water. In (g) the increase in output clusters have given the buffalo better contrast.

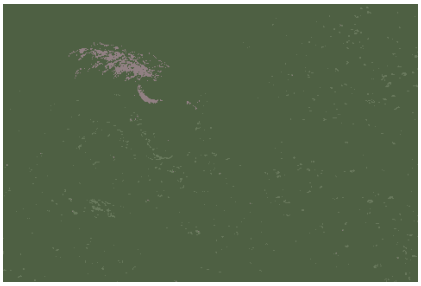
(a) Original image.

(b) 77 partitions.



(c) KPCA 2 clusters.

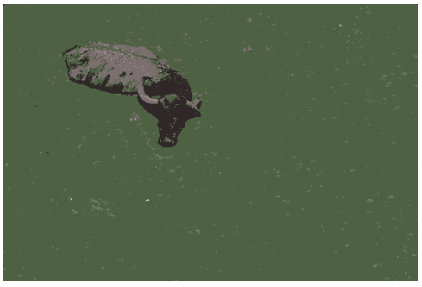
(d) KECA 2 clusters.



(e) KPCA 4 clusters.

(f) KECA 4 clusters.

(g) KPCA 10 clusters.



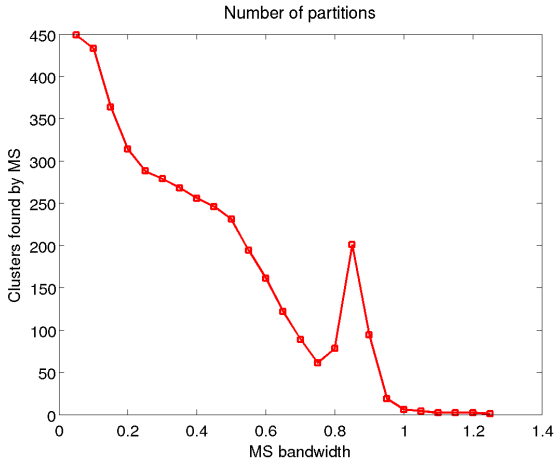
(h) KECA 10 clusters.

Figure 6.16: Swiming water buffalo.

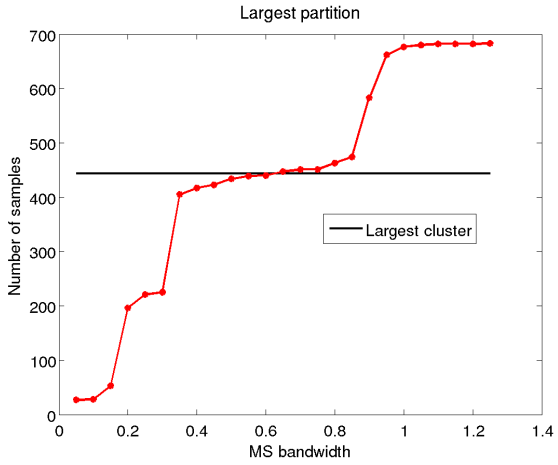
6.3 Additional Datasets

6.3.1 Wisconsin Breast Cancer Dataset

The Wisconsin breast cancer dataset contains 683 samples of clinical cases of breast cancer diagnosis with each sample consisting of nine different attributes. It can be found in [11] where it was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. The dataset and a multi surface method of pattern separation was presented in [43].



(a) Number of partitions.



(b) Largest partition.

Figure 6.17: Mean shift result.

Figure 6.17a shows the number of clusters found by using 100 mean shift iterations without blur. We notice an increase in the number of partitions

found when the bandwidth reaches a certain size, here for $h_{\text{ms}} = 0.75$, before it drops again. This was also seen for the Iris dataset in Fig. 6.1 and for the cows picture in Fig. 6.8 and is caused by the increasing kernel size causing modes to merge.

In Fig. 6.17b the number of samples in the largest partition. The horizontal black line shows the number of samples in the larges class (benign). For every sample the size of the largest partition is over this line, the *minimum* error clustering error increases by one. This means that for $h_{\text{ms}} = 1.0$, over the final clustering will contain a minimum of 150 clustering errors.

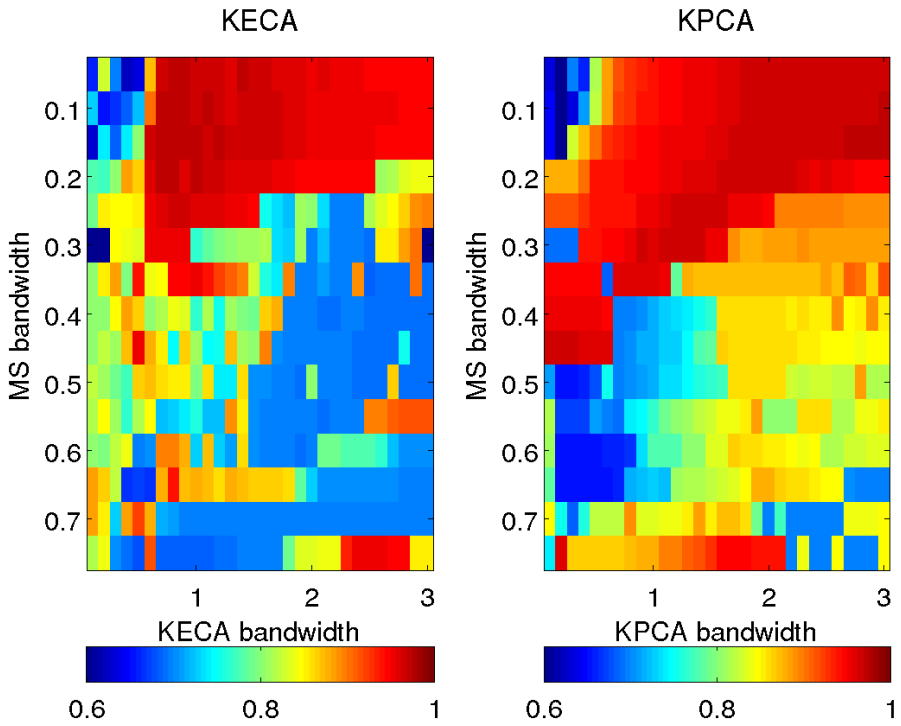


Figure 6.18: Clustering accuracy.

Figure 6.18 shows the accuracy for MSSC as a function of mean shift bandwidth along the vertical axis and spectral bandwidth along the horizontal axis. KECA with a cosine distance for k-means is used to obtain the result in the left subplot while the right is based on centered KPCA with an Euclidean distance in k-means.

We see that the best results when using KECA is when the mean shift bandwidth is small and the spectral bandwidth is around $h_{\text{KECA}} \approx 1.0$. The highest accuracy is obtained with $h_{\text{ms}} = 0.05$ and $h_{\text{KECA}} = 0.9$ when 664 of

the 683 samples are correctly classified, which corresponds to an accuracy of 97.2%. For KPCA the highest accuracy is 96.9% (662 samples) obtained with $h_{\text{ms}} \in \{0.05, 0.10\}$ for $h_{\text{KPCA}} \in \{2.7, 2.8, 2.9\}$.

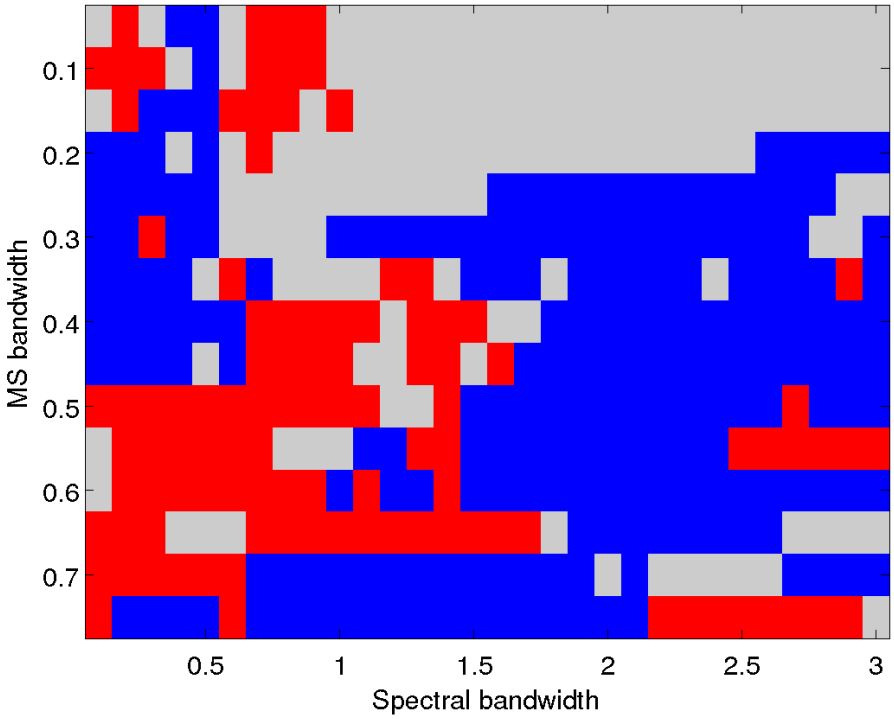


Figure 6.19: Difference in clustering accuracy.

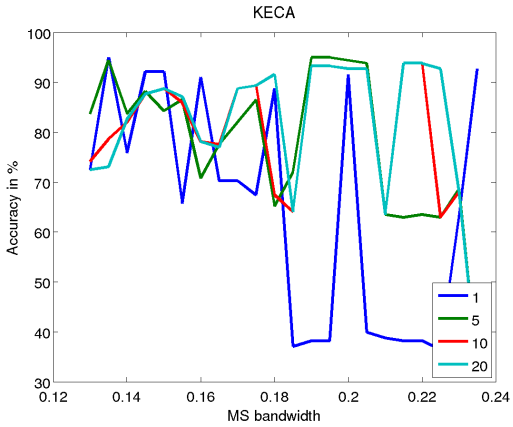
Figure 6.19 shows the difference in the clustering accuracy in Fig. 6.18. The gray area are represents where neither method is better than the other by more than 2.5%. The blue area is where KPCA has a significaly higher classification accuracy than KECA, while the oposite is true for the red areas.

Combined with Fig. 6.18 we can say that both methods have highest accuracy when the mean shift kernel size is small. KPCA has a more stable accuracy than KECA for this dataset, but when overall best accuracy is obtained with KECA.

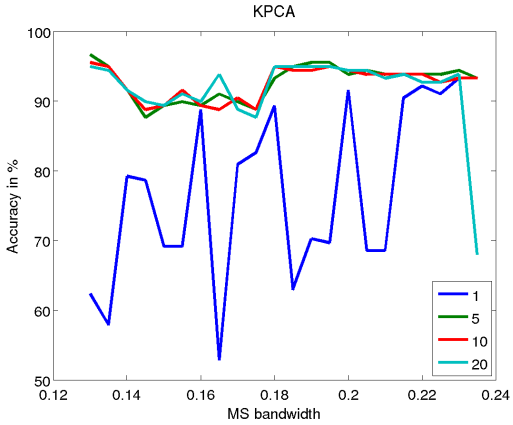
6.3.2 Wine Dataset

The wine dataset consists of $l = 13$ chemical features from $n = 178$ different Italian wines. Each wine comes from one of three different cultivars, which

we seek to cluster correctly. The dataset was obtained from UCI repository [11].



(a) KECA.



(b) KPCA.

Figure 6.20: Accuracy.

In Fig. 6.20 the clustering accuracy is plotted as a function of the mean shift and spectral bandwidths. The spectral bandwidths are chosen to be some factor times the mean shift bandwidth, the different factors are 1, 5, 10 and 20. These are represented by different coloured lines in (a) and (b) where KECA and KPCA are used in the second stage of MSSC.

The blue curve in both plots resembles the result when the spectral bandwidth was set equal to the mean shift bandwidth for the Iris data in Fig. 6.3. We also notice that the result using KPCA appears to be more stable as a function of kernel size, as was seen in Fig. 6.18. The maximum clustering accuracy for KECA was 94.9% and 96.6% for KPCA.

Chapter 7

Conclusion

In this thesis we have presented novel Mean Shift Spectral Clustering (MSSC), a clustering procedure which allows the use of spectral clustering methods on large datasets. This two-stage approach enabled us to segment images with as many feature vectors as pixels. MSSC also proved to give a better clustering result than using spectral methods with the same kernel size directly for a non-linear toy data example.

The mean shift algorithm provides an initial clustering of the dataset by grouping points closest to the same local mode in the probability density estimate together. Since this estimate is obtained by Parzen windowing, a non-parametric density estimator, these clusters represent a natural partitioning of the dataset. Mean shift can find partitions separated by non-linear boundaries, which makes it well suited for preprocessing the data for spectral clustering. Thus MSSC can be used to find non-linear cluster structures in large datasets.

We showed how the number of partitions found by the mean shift algorithm varied as a function of the kernel size used. While this number generally decreases with increasing kernel size, we showed that with a fixed number of iterations, the number of partitions can increase again. This is caused by the mean shift algorithm moving the mode-finding vectors in small steps in dense regions. The merging of dense regions with increased kernel size might therefore require a large number of iterations to converge. To speed up convergence and hence provide a more predictable number of output partitions we used blurring mean shift.

In order to perform the second stage of the clustering procedure, regarding the spectral clustering based on the output of the stage one mean shift procedure, we introduced the recent Kernel Entropy Component Analysis (KECA) method. Comparisons were made against the more well known Kernel Principal Component Analysis method (KPCA). Results indicate that

KECA achieved the highest clustering accuracy and selected more relevant features in an image segmentation example. KPCA generally had a more stable performance when varying the kernel sizes. For both cases, the MSSC procedure gave good clustering results.

7.1 Suggestions for Further Work

7.1.1 General Suggestions

This section contains some general observations on how to further the development of novel MSSC.

Further testing. Apply the novel MSSC procedure to challenging data sets exhibiting complex structure within fields such as bioinformatics, neuroinformatics, remote sensing, etc.

Kernel size selection. Develop general procedures for automated kernel size selection in both stage one and stage two.

Using other kernels. In this thesis the Gaussian kernel was used for all experiments, both in mean shift and the spectral clustering. There are a variety of different kernels that could be used. Some other alternatives can be found in [26].

Three-stage approach. The mean shift algorithm represents an computational bottleneck for MSSC [28]. It is possible to use a simpler clustering algorithm, for instance k-means, to reduce the number of feature vectors input to mean shift. In the mean shift algorithm, information about the k-means pre-partitioning can be included by weighting these pre-partitions by the number of points assigned to it. The idea of gradually more complicated methods can be extended to use even more stages to handle particularly large clustering tasks.

Preprocessing images. The image segmentation performed in this thesis used the pixel intensities as features without any normalization while the x and y coordinates were scaled to a be a fraction of the pixel intensity range. There are many different ways to implement pre- and post processing of image clustering that can improve the segmentation result. A tutorial can be found in [40].

Clustering new data. A way to assign new feature vectors to clusters found by MSSC is to find out which mean shift mode these feature

vectors converge to. This by using mean shift with density estimate based on the original dataset. If the new feature vectors are not included in the data used for Parzen windowing, they will to converge to one of the mean shift modes unless some very unlikely symmetry causes it to remain stationary. New feature vectors should not simply be assigned to the closest mode in case the dataset has a non-linear cluster structure.

7.1.2 First Stage

As the mean shift algorithm is a computational bottleneck in MSSC, this section's primary focus is how to reduce the computational load in the first stage.

K-means in first stage. It was mentioned in [28] that k-means or other simple clustering methods such as Expectation Maximization with Gaussian mixture models (see Section 2.3) can be used to obtain the initial partitioning.

Reducing the number of points for density estimate. It is possible to use only the closest feature vectors[1] to the position mode-finding vector to obtain the density estimate. The number of feature vectors used for each mode-finding vector can be significantly reduced, depending on how much the feature vectors have to contribute to the density estimate to avoid being cut. This option was included in the code developed for this thesis, but not utilized as it caused memory problems for large datasets. It is possible that better implementation of this system can reduce the computational load of mean shift.

Increase in number of partitions. It is possible that the sudden increase in number of partitions despite increasing kernel size can be used to determine a good kernel size for the system. This can be an indication that the algorithm starts describing the global structure of the data, which means that the bandwidth is too large as this should be done by the spectral method in MSSC. One could gradually increase the number of mean shift iterations and see how long increases in cluster numbers last as a function of mean shift iterations. This can be used to perform an analysis analog to what is done for dendograms [39, Chapter 13].

Varying bandwidth between mean shift iterations. Gradually reducing the kernel size may speed up convergence as the mean shift moves

[1] Mode-finding vectors from the previous iteration when using blur.

points further in regions of low densities. However one must study how this influcences the clustering result.

Using fewer mode-finding vectors than data points. For datasets with many low-dimensional samples, one might define a grid of mode-finding vectors and use the mean shift algorithm on these. The density estimate would still be based on the full dataset, but if the grid contains fewer points than the dataset this will reduce the computational load. Determining which data points converge to the same mode can be done by some form of nearest neighbour assignment based on the points in this grid. For this to be a good approach, the dimension of the dataset should be low.

7.1.3 Second Stage

This section lists some suggestion for further development related to the second stage.

Laplacian eigenmaps in MSSC. Laplacian eigenmaps will generally give different results than KECA and KPCA and should therefore be tested in MSSC.

Dimension of transformed data. All clustering results in this thesis are based on setting the dimension of the transformed data equal to the number of output clusters specified. This to avoid a tedious optimization process by including an additional parameter to vary. However, increasing the dimension of the transformed data might improve the performance of MSSC in some cases.

KECA sensitivity to kernel size. The result of MSSC with KECA showed a tendency to vary more than with KPCA. This could be investigated. It might be connected with the cosine metric in the k-means algorithm used in KECA spectral clustering.

Divergence measure for kernel matrix. The kernel matrix was constructed based on the Cauchy-Schwarz divergence between partitions. Other divergence measures could also be used.

Out of sample projection of outliers. In the experiments conducted for this thesis, the mean shift algorithm often found several one point partitions for small bandwidths. These are included in the partition kernel matrix, and contribute to its size and eigendecomposition result. To avoid the outliers influencing the result, one can form the kernel matrix

based on the other partitions and treat the outlier partitions as out-of-sample data and project them feature space principal axes found by the spectral method. An example of projecting out-of-sample data points is included in [18].

List of Figures

1.1 Two datasets. 2
1.2 K-means solution. 3
1.3 MSSC solution. 4

2.1 Dataset with two different initial sets of cluster representatives. 20
2.2 Convergence after 6 iterations. 21
2.3 Convergance after 5 iterations. 22
2.4 Cost function and number of cluster changes. 23
2.5 Dataset with three normal components. 32
2.6 Gaussian mixtures fitted by EM. 33
2.7 Dataset with mixture contours. 34
2.8 KDE with $h = 0.5$ and mean shift. 41
2.9 KDE with $h = 1$ and mean shift. 42
2.10 KDE with $h = 0.5$ after one "blurring". 44
2.11 $h = 0.5$, blurring and non-blurring mean shift. 45
2.12 Two concentric circles with noise. 54
2.13 Circle and "reverse L" shape. 55
2.14 Projection onto first 3 components. 56
2.15 Neighbouring points example. 58
2.16 Ring data. 63
2.17 First two components of transformation. 64
2.18 Three groups of data. 66
2.19 Four different eigenmaps. 67
2.20 kNN with $k = 25$. 68

3.1 Speed of molecules in a gas. 78

4.1 Two rings and a dense cluster. 87
4.2 Eigenvectors. 88
4.3 Clustering result. 90
4.4 Transformed data. 91

4.5 Increasing bandwidth to $h = 4.5$. 92

6.1 Number of clusters for different mean shift bandwidths. 102
6.2 Number of clusters found by mean shift, different number of iterations. 103
6.3 Classification accuracy as a function of mean shift bandwidth. 104
6.4 Classification accuracy as a function of spectral bandwidth. . . 106
6.5 KECA and KPCA classification, Euclidean distance used for spectral step. 107
6.6 KECA and KPCA classification, cosine distance used for spectral step. 108
6.7 The black and white cows picture. 109
6.8 Number of clusters found by mean shift. 111
6.9 Four cluster results for KECA and KPCA. 114
6.10 Larger pixel coordinate range. 115
6.11 Comparison with k-means. 116
6.12 Plane. 117
6.13 Spectral clustering with $h = 2.0$ 118
6.14 Clustering results. 119
6.15 Plane. 120
6.16 Swiming water buffalo. 122
6.17 Mean shift result. 123
6.18 Clustering accuracy. 124
6.19 Difference in clustering accuracy. 125
6.20 Accuracy. 126

Bibliography

[1] S. Alpert, M. Galun, R. Basri, and A. Brandt. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, June 2007.

[2] H. Anton and C. Rorres. *Elementary Linear Algebra : Applications Version*. Wiley, New York, 2005.

[3] G. Arimond and A. Elfessi. A clustering method for categorical data in tourism market segmentation research. *Journal of Travel Research*, 39(4):391–397, 2001.

[4] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[5] Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[6] Y. Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.

[7] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(5):603–619, 2002.

[8] D. Erdogmus and J.C. Principe. From linear adaptive filtering to nonlinear information processing - the design and analysis of information processing systems. *Signal Processing Magazine, IEEE*, 23(6):14–33, 2006.

[9] M. Fashing and C. Tomasi. Mean shift is a bound optimization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(3):471–474, 2005.

[10] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Human Genetics*, 7(2):179–188, 1936.

[11] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[12] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975.

[13] D. Giancoli. *Physics : Principles With Applications*. Pearson/Prentice Hall, Upper Saddle River, N.J, 2005.

[14] M. Girolami. Orthogonal series density estimation and the kernel eigenvalue problem. *Neural Computation*, 14(3):669–688, 2002.

[15] R. Hartley. Transmission of information. *The Bell System Technical Journal*, 1928.

[16] A.K. Jain. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[17] E.T. Jaynes. Information theory and statistical mechanics. *Statistical Physics. Brandeis Lectures*, 3:160–185, 1963.

[18] R. Jenssen. Kernel entropy component analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(5):847–860, 2010.

[19] R. Jenssen and T. Eltoft. A new information theoretic analysis of sum-of-squared-error kernel clustering. *Neurocomputing*, 72(1-3):23–31, 2008.

[20] R. Johnson and D. Wichern. *Applied Multivariate Statistical Analysis*. Pearson Prentice Hall, Upper Saddle River, N.J, 2007.

[21] J. Kapur. *Measures of Information and Their Applications*. Wiley, New York, 1994.

[22] T. Lillesand, R. Kiefer, and J. Chipman. *Remote Sensing and Image Interpretation*. Wiley, New York, 2004.

[23] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proc. 8th Int'l Conf. Computer Vision*, volume 2, pages 416–423, July 2001.

[24] G.J. McLachlan and D. Peel. *Finite Mixture Models*, volume 299. Wiley-Interscience, 2000.

[25] S. Mika, B. Schölkopf, A.J. Smola, K.R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. *Advances in neural information processing systems*, 11(1):536–542, 1999.

[26] K.R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *Neural Networks, IEEE Transactions on*, 12(2):181–201, 2001.

[27] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002.

[28] U. Ozertem, D. Erdogmus, and R. Jenssen. Mean shift spectral clustering. *Pattern Recognition*, 41(6):1924–1938, 2008.

[29] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 727–734. San Francisco, 2000.

[30] J.C. Principe. *Information Theoretic Learning : Renyi's Entropy and Kernel Perspectives*. Springer, New York, 2010.

[31] S. Rao, A. de Medeiros Martins, and J.C. Principe. Mean shift: An information theoretic perspective. *Pattern Recognition Letters*, 30(3):222–230, 2009.

[32] A. Renyi. On measures of entropy and information. *Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pages 547–561, 1961.

[33] L. Scharf. *Statistical Signal Processing : Detection, Estimation, and Time Series Analysis*. Addison-Wesley Pub. Co, Reading, Mass, 1991.

[34] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[35] C.E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(1):379–423, 623–656, 1948.

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

[37] K.S. Song. Renyi information, loglikelihood and an intrinsic distribution measure. *Journal of Statistical Planning and Inference*, 93(1-2):51–69, 2001.

[38] A. Strehl, J. Ghosh, and R. Mooney. Impact of similarity measures on web-page clustering. In *Workshop on Artificial Intelligence for Web Search (AAAI 2000)*, pages 58–64, 2000.

[39] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Academic Press, 2008.

[40] T.N. Tran, R. Wehrens, and L. Buydens. Clustering multispectral images: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 77(1):3–17, 2005.

[41] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007.

[42] R.E. Walpole, R.H. Myers, S.L. Myers, and K. Ye. *Probability & statistics for engineers & scientists*. Pearson Prentice Hall, Upper Saddle River, NJ, 2007.

[43] W.H. Wolberg and O.L. Mangasarian. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23):9193, 1990.

[44] K.Y. Yeung, C. Fraley, A. Murua, A.E. Raftery, and W.L. Ruzzo. Model-based clustering and data transformations for gene expression data. *Bioinformatics*, 17(10):977–987, 2001.

[45] S. Zhang, R.S. Wang, and X.S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A : Statistical Mechanics and its Applications*, 374(1):483–490, 2007.


UNIVERSITY
OF TROMSØ