An exploration of cluster analysis and its practical applications

Thomas Porter

October 27, 2022

Abstract

This report provides an introduction to cluster analysis along with an example of its application on the MNIST dataset: a collection of 70,000 images of handwritten digits from 0-9. We discuss three methods of clustering: K-means, OPTICS and the Gaussian mixture model (and Expectation Maximisation algorithm) before describing principal component analysis, a common method used for data compression. These methods are subsequently applied to the MNIST dataset with their efficacy being evaluated using a number of both internal and external methods.

This piece of work is a result of my own work except where it forms an assessment based on group project work. In the case of a group project, the work has been prepared in collaboration with other members of the group. Material from the work of others not involved in the project has been acknowledged and quotations and paraphrases suitably indicated.

Contents

Chapter 1

Introduction

1.1 Cluster Analysis

Cluster Analysis or *clustering* is an unsupervised learning technique which is used to organise observations into different groups based on their features in such a way that observations within the same cluster have features more similar to each other than to those in different clusters. It is a very useful technique for data organisation and for learning relationships between different observations. As an unsupervised learning technique, it is performed on unlabelled data making it a very valuable method for data analysis in the real world since most of the data found there is unlabelled. Indeed, cluster analysis is used in many fields such as pattern recognition (Saha et al. 2019) and bioinformatics (Abu-Jamous et al. 2015). Cluster analysis dates back to 1932 with its first use coming in anthropology (Driver & Kroeber 1932). It was also used a few years later in psychology (Zubin 1938, Tryon 1939, Cattell 1943). The methods used in the early stages of its development were vastly different to those of the modern age. This is mainly down to the fact that these methods pre-date the age of modern computing (Aggarwal & Reddy 2014). Almost all modern methods are optimized for computational application with the best clusterings often being those with both high accuracy and low computational cost. Figure 1.1 illustrates a toy example of a dataset which has been clustered with the different colours representing different cluster assignments.

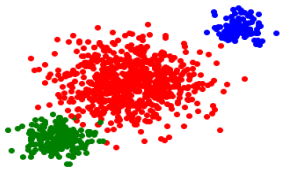


Figure 1.1: A visualisation of a two-dimensional dataset that has been clustered.

1.2 Notation

We will be considering datasets of the form $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_n)^T$ where $\mathbf{x}_j \in \mathbb{R}^p$. Each $\mathbf{x}_j$ is known as an observation and within each observation there are p features. We shall therefore denote x_{jl} as the l^{th} feature of the j^{th} observation. In this report we will purely be discussing quantitative features.

1.3 Types of clustering

There are several different types of methods used for clustering. These methods are primarily separated by their different definitions of exactly what a cluster is. They also differ in that different steps are taken in finding these clusters. The following are some of the most common types of clustering models.

1.3.1 Distance-based methods

Distance-based models define clusters as groups of observations with minimal "distance" between them and maximal "distance" between them and points in other clusters. This distance can be measured in a variety of ways depending on the type of data being clustered. The most common distance function used is the Euclidean distance function, $d_{Euclidean}$ which is defined later on in this chapter. There are two main types of distance-based method; centroid-based and hierarchical. They are primarily separated by the fact that in centroid-based models, clusters are represented by a "centroid", a vector which summarises the observations in its cluster. Centroid-based models usually result in just one clustering whereas hierarchical models provide multiple clusterings. They are also different in that the person undertaking the clustering must provide the number of clusters they want the data to be separated into for centroid-based models whereas hierarchical models provide clusterings with all possible numbers of clusters.

Centroid-based clustering

In centroid-based clustering, each cluster is represented by a central vector or "centroid". Every observation in the dataset can then be allocated to a cluster by determining which centroid the observation is closest to. There are a number of distance functions which one can use to measure this closeness, the most common one being Euclidean distance. The choice of central vector and distance function is crucial in determining the final clustering (Aggarwal & Reddy 2014). Centroid-based models tend to work iteratively by assigning each observation to its nearest centroid before updating the position of the centroid based on the points assigned to it. An example of a centroid-based method is the k-means algorithm. The general idea of the algorithm is as follows (James et al. 2013).

1. Randomly assign an integer from {1,...,k} to each observation to form the initial cluster assignments.

2. Calculate the mean vector for each cluster based on the observations assigned to it.

3. Go through the observations assigning them to the cluster with the closest mean vector.

Steps 2 and 3 are repeated iteratively until the assigned clusters no longer change. This is a very brief introduction to the k-means algorithm which we will be discussing in much greater detail in Chapter 2. Figure 1.2 shows a toy dataset which has been clustered using the k-means algorithm.

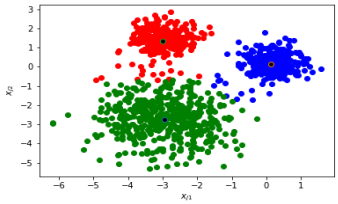


Figure 1.2: A visualisation of a two-dimensional dataset that has been clustered using the k-means algorithm. The black dots denote the cluster centroids.

Hierarchical clustering

Hierarchical clustering involves the formation of clusters which have an overall ordering. These clusterings are represented by a dendrogram (Schubert 2017) which provides an entire system of clusterings which merge together at certain distances. This is in stark contrast to centroid-based models which provide a single clustering. Depending on how many clusters we wish to make, we can simply observe the level of the dendrogram at which that many clusters are present. An example of a hierarchical clustering model is agglomerative clustering (Sneath 1957). This is a bottom-up approach to clustering which begins with all observations in their own clusters at the bottom. The algorithm then works iteratively merging the closest clusters one at a time. Like centroid based clustering the end results depend massively on the distance function employed. However they also depend on the method employed for determining the distance between clusters and thus which clusters are to be merged. These are known as *linkages*. Examples include *simple linkage* in which the distance between two clusters is the smallest distance between any two points in the two clusters, *complete linkage* where the distance is given by the maximal distance between any two points in the two clusters and *average linkage* which as the name suggests takes the distance between all points in the two clusters and returns the average (mean) (Murphy 2012). There are advantages and disadvantages to each of these and so the choice of which one to use depends on the data being clustered. For example, two clusters could be extremely close to each other in terms of average linkage. But if there is even one outlier (an observation that differs massively from the rest of the dataset) then the two clusters may not be merged if complete linkage were

to be used. Therefore the type of linkage has to be carefully considered before using agglomerative clustering. Figure 1.3 displays an example of a dendrogram which can be used to visualise the results of agglomerative clustering.

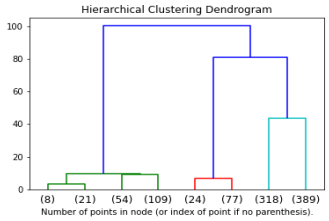


Figure 1.3: The top 2 levels of a dendrogram illustrating agglomerative clustering of the data in Figure 1.1

1.3.2 Density-based methods

In density-based clustering, a cluster is defined as an area of high density surrounded by an area of lower density (Schubert 2017). Observations that are contained in areas of low density tend to be classified as *outliers* or *noise*. The majority of density-based clustering models have two key parameters which determine whether a given observation is classed as a *core point*, *border point* or *outlier*. These are $\epsilon \in \mathbb{R}_{\geq 0}$ and $MinPts \in \mathbb{N}$. An observation, $\mathbf{x}_j$, is a *core point* if there are at least $MinPts$ points (including $\mathbf{x}_j$) contained in an ϵ-neighbourhood of $\mathbf{x}_j$. Like distance-based methods, any distance function can be used. An observation, $\mathbf{x}_j$ is a *border point* if it is in an ϵ-neighbourhood of a core point but there are fewer than $MinPts$ points within the neighbourhood. Lastly an observation, $\mathbf{x}_j$ is an *outlier* if it is not a core point and is not contained in the ϵ-neighbourhood of a core point. Once we decide on the values for ϵ and $MinPts$ the method of clustering is simple. Density-based spatial clustering of applications with noise (DBSCAN) is a commonly-used density-based clustering algorithm. Its basic steps are displayed below (Ester et al. 1996):

1. Randomly choose an observation, $\mathbf{x}_j$ as our starting point. If it is not a core point, mark it as an outlier and repeat.
2. Begin a new cluster C_k
3. For each point $\mathbf{x}_i$ in the ϵ-neighbourhood of $\mathbf{x}_j$ (including $\mathbf{x}_j$):
   (a) If $\mathbf{x}_i$ is already assigned to a cluster, move onto the next point.
   (b) Assign $\mathbf{x}_i$ to cluster C_k.
   (c) If $\mathbf{x}_i$ is a core point, repeat step 3 for the points in the ϵ-neighbourhood of $\mathbf{x}_i$
4. Once all neighbours have been labelled then the cluster is complete.
5. Repeat until all observations have been labelled.

The main advantage of density based methods over distance-based methods is that they can cluster irregularly-shaped clusters such as those shown in Figure 1.4 whereas distance based methods usually tend to only be able to cluster spherical clusters. This does of course usually depend on the distance function used.

Figure 1.4: A visualisation of a two-dimensional irregularly-shaped dataset that has been clustered.

1.3.3 Probabilistic models

In probabilistic clustering, the clusters are defined as groups of observations which are most likely to come from the same distribution. Having assumed these distributions, one must estimate their optimal parameters using the expectation maximization (EM) algorithm. These optimal parameters are estimated using the dataset in such a way that they have maximum likelihood fit to the assumed distributions (Aggarwal & Reddy 2014). One can then go through each data point and calculate the probability of it belonging to each cluster using each cluster's estimated parameters. Those points which are most likely to be in a certain cluster will have a higher probability, whilst outliers will have a low probability. Each observation is assigned to the cluster which maximises this probability. One disadvantage of probabilistic models is that one must assume that the data belong to a number of probabilistic distributions, which they may not.

1.4 Distance functions

So far in this report there have been a number of references to distance functions and so we shall now define them before giving a few examples of distance functions which are commonly used in distance-based and density-based clustering methods.

Definition 1.4.1 (Distance function)**.** A *distance function* on a set $\mathbf{x}$ is defined as a function $d : \mathbf{x} \times \mathbf{x} \to \mathbb{R}_{\geq 0}$ satisfying the following properties (Schubert 2017):

1. $d(\mathbf{x}_j, \mathbf{x}_l) = 0 \Leftrightarrow \mathbf{x}_j = \mathbf{x}_l$
2. $d(\mathbf{x}_j, \mathbf{x}_l) = d(\mathbf{x}_l, \mathbf{x}_j)$
3. $d(\mathbf{x}_j, \mathbf{x}_l) \leq d(\mathbf{x}_j, \mathbf{x}_m) + d(\mathbf{x}_m, \mathbf{x}_l)$
4. $d(\mathbf{x}_j, \mathbf{x}_l) \geq 0$

where $\mathbf{x}_j, \mathbf{x}_l, \mathbf{x}_m \in \mathbf{x}$ and the last property is a result of properties 1,2 and 3.

The following are some examples of commonly used distance functions.

- $d_{Euclidean}(\mathbf{x}_j, \mathbf{x}_l) = \sqrt{\sum_{m=1}^{q} (x_{jm} - x_{lm})^2} = \|\mathbf{x}_j - \mathbf{x}_l\|_2$
- $d_{Manhattan}(\mathbf{x}_j, \mathbf{x}_l) = \sum_{m=1}^{q} |x_{jm} - x_{lm}| = \|\mathbf{x}_j - \mathbf{x}_l\|_1$
- These are special cases of the Minkowski or L^p norm:
  $d_{L^p}(\mathbf{x}_j, \mathbf{x}_l) = (\sum_{m=1}^{q} (x_{jm} - x_{lm})^p)^{\frac{1}{p}} = \|\mathbf{x}_j - \mathbf{x}_l\|_p$
- $d_{Chebyshev}(\mathbf{x}_j, \mathbf{x}_l) = \max_{1 \leq m \leq q} (|x_{jm} - x_{lm}|) = \lim_{p \to \infty} \|\mathbf{x}_j - \mathbf{x}_l\|_p$

Note that above we have briefly changed the notation for the number of features in an observation from p to q to avoid confusion with the p in the L^p norm.

1.5 Overview of the remaining chapters

In the next three chapters we will discuss three types of clustering model in detail. These are the k-means algorithm, the OPTICS algorithm and the Gaussian mixture model (and Expectation Maximisation algorithm) which are a distance-based method, density-based method and probabilistic method respectively. We will then move onto describing some of the methods of evaluating the results of cluster analysis in Chapter 5. Evaluation methods can be split into two main categories: internal and external. Internal methods do not make use of the true cluster labels whereas external methods do. Specifically we will discuss the internal methods: Silhouette coefficient and Davies-Bouldin index and the external methods: Purity and Rand index. We will have a brief look at the three clustering methods in action in Chapter 6. We will see how they perform on four toy datasets in order to establish the advantages and disadvantages of each method for certain types of data. In Chapter 7 we will discuss principal component analysis (PCA) which is a method of dimensionality reduction. This is a very useful tool for improving the efficiency of a clustering model since it reduces the number of features that the cluster analysis is performed on. However it still retains most of the variability of the data since it calculates the linear combinations of the features which maximise the variance in the data. The variance of the data represents the meaningful information that can be extracted from the dataset and so one can still draw useful conclusions from the results of cluster analysis on PCA-reduced data. In Chapter 8 we will look at the MNIST dataset which is one of the most widely used datasets for cluster analysis in the real world. It is a collection of 70,000 images of handwritten digits from 0-9. Since the dataset consists of images we will first have to explain how to pre-process the data so that it can be clustered. We will then perform PCA on it to make it more efficient to cluster before applying our clustering methods from Chapters 2,3 and 4. Lastly we will evaluate the results using the methods described in Chapter 5. Chapter 9 concludes this report providing a summary of everything covered as well as providing some insight into what the future of cluster analysis holds.

Chapter 2

K-means clustering

2.1 Background

The k-means algorithm is an example of a distance-based clustering method. Whilst many similar algorithms were proposed throughout the 1950's and 1960's, the first use of the name "k-means" came in 1967 from James MacQueen, of the Department of Statistics of the University of California (Macqueen 1967). Before running the algorithm one must first specify the number of clusters, K. Let $C_1, ..., C_K$ denote the sets containing the indices of the observations in each cluster. These sets should satisfy the following properties (James et al. 2013):

1. $C_1 \cup C_2 \cup ... \cup C_K = \{1, 2, ..., n\}$
2. $C_k \cap C_{k'} = \emptyset \ \ \forall \ k \neq k'$

In other words, each observation from the dataset should be assigned to exactly one cluster. The main objective when clustering is to minimise within-cluster-variation (James et al. 2013).

Definition 2.1.1 (Within-cluster-variation)**.** The *within-cluster-variation* of a cluster C_k is the degree to which the observations in the cluster are different from each other. We shall denote it as,

$$W(C_k) = \frac{1}{|C_k|} \sum_{j,j' \in C_k} \sum_{l=1}^{p} \left(x_{jl} - x_{j'l}\right)^2 \tag{2.1}$$

Note that $|C_k|$ denotes the number of observations in cluster k. Note also that we have used squared Euclidean distance for reasons that will be explained later. The task of k-means clustering then comes down to minimising the within-cluster distance over all K clusters, i.e.

$$\underset{C_1,...,C_K}{\text{minimize}} \left\{ \sum_{k=1}^{K} W(C_k) \right\} \tag{2.2}$$

There are $\left\{ {n \atop k} \right\} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n$ ways to organise n observations into K clusters hence finding a global minimum for (2.2) is almost impossible (Lengyel 1994). For example there are roughly 2.75×10^{93} ways to organise 100 observations into 10 clusters. Luckily, Algorithm 1 provides a local optimum to (2.2) (Lloyd 1982).

Algorithm 1: LLOYD'S ALGORITHM

Input: A dataset $\mathbf{x}$ and the number of clusters K.

Output: The partitioning of $\mathbf{x}$ into K clusters.

1 Randomly choose K points from $(\mathbf{x}_1, ..., \mathbf{x}_n)^T$ as the initial cluster centroids. Denote these as $(\boldsymbol{\mu}_1, ..., \boldsymbol{\mu}_K)$

2 **repeat**

3 Assign each observation to the closest centroid using Euclidean distance, more specifically, for each $\mathbf{x}_j$ find d_M and assign $\mathbf{x}_j$ to the cluster with centroid $\boldsymbol{\mu}_M$ where

$$d_M = min\{d_1, ..., d_K\} \quad \text{and} \quad d_k = \sqrt{\sum_{l=1}^{p}(x_{jl} - \mu_{kl})^2} \tag{2.3}$$

4 Update each centroid as the mean vector of the observations in its cluster, more specifically, for each $\boldsymbol{\mu}_k$,

$$\mu_{kl}^{\text{new}} = \frac{1}{|C_k|}\sum_{j \in C_k} x_{jl} \tag{2.4}$$

5 **until** *the cluster assignments no longer change*

6 **return** *clusters*

Steps 3 and 4 are repeated iteratively until the cluster assignments no longer change. At this stage a local optimum has been reached. However, one must note the use of the word local and that the quality of any clustering is highly sensitive to the choice of initial cluster centroids. In an ideal world, each initial cluster centroid will come from a different cluster. However, the probability of this happening is very small (Schubert 2017). Assuming that each cluster is of the same size, $\frac{n}{k}$, then the probability of each initial centroid belonging to a different cluster is equal to

$$\frac{\text{no. of ways to choose one centroid from each cluster}}{\text{no. of ways to choose k centroids}} = \frac{k!\left(\frac{n}{k}\right)^k}{\left((k)\left(\frac{n}{k}\right)\right)^k} = \frac{k!}{k^k} \tag{2.5}$$

Lets say there are k=6 clusters, the probability of randomly choosing cluster centroids from six different clusters is equal to $\frac{5}{324} \approx 0.0154$. Therefore it is often common practice to execute the algorithm a number of times so that a range of different centroid

initializations can be used and then the best clustering can be chosen. To find the best clustering, one could choose the clustering for which (2.2) is the smallest. Alternatively, one could use one of a range of evaluation methods which will be discussed in Chapter 7. Another way of finding something closer to a global optimum would be to simply take the mean centroid positions across all repetitions and run the algorithm using those mean cluster centroids as the initial choice of centroids instead of choosing them at random. How do we know that Lloyd's algorithm is guaranteed to decrease (2.2) at each step? The answer becomes clearer thanks to the following identity (James et al. 2013).

$$\frac{1}{|C_k|}\sum_{j,j'\in C_k}\sum_{l=1}^{p}\left(x_{jl}-x_{j'l}\right)^2 = 2\sum_{j\in C_k}\sum_{l=1}^{p}(x_{jl}-\mu_{kl}^{\text{new}})^2 \tag{2.6}$$

where μ_{kl}^{new} is the mean for feature l of the observations in C_k as defined in (2.4). Our chosen definition for the updated cluster centroid $\boldsymbol{\mu}_k$ as the mean vector of the observations in cluster C_k minimises the sum-of-squared deviations. Note also that assigning each observation to the cluster with the closest centroid in step 3 can only decrease (2.6). Therefore each iteration of steps 3 and 4 can only improve (2.6).

2.2 Choosing the value of K

One of the first problems that one encounters when performing k-means clustering is the question of choosing the number of clusters, K. This can be particularly challenging when we are unfamiliar with the dataset or it cannot easily be visualised due to being high-dimensional (i.e. p is large). It is trivial that increasing the number of clusters will improve the clustering since observations will be closer to their centroid. Indeed if $K = n$, each observation will have its own cluster hence the within-cluster-variation would have value 0. Therefore it is about finding a balance between an accurate clustering and small K. Luckily there is a heuristic technique for finding the optimal value of K.

The *elbow method* involves plotting the proportion of variance explained (PVE) against the number of clusters K (Raschka 2015). There are a number of ways that one can measure PVE but the most commonly used is the distortion score.

Definition 2.2.1 (Distortion)**.** The *distortion score* for a clustering $C = \{C_1, ..., C_K\}$ is defined as

$$\text{Distortion}(C) = \sum_{k=1}^{K}\sum_{j\in C_k}\sum_{l=1}^{p}(x_{jl}-\mu_{kl}^{\text{new}})^2 \tag{2.7}$$

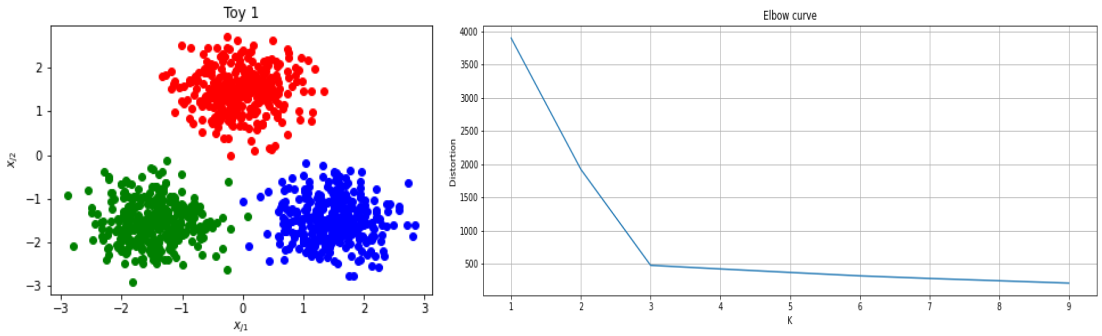
Using identity (2.6) we can see that the distortion is related to the within-cluster variance by the following equation

$$\text{Distortion}(C) = \sum_{k=1}^{K}\frac{1}{2}\frac{1}{|C_k|}\sum_{j,j'\in C_k}\sum_{l=1}^{p}\left(x_{jl}-x_{j'l}\right)^2 = \frac{1}{2}\sum_{k=1}^{K}W(C_k) \tag{2.8}$$

Therefore optimising objective (2.2) is equivalent to minimising the distortion. Once we have plotted the distortion against K, the optimum value of K is located at the "elbow point". If plotted correctly, the plot should show distortion decreasing as K increases. The distortion should decrease more sharply at the start before beginning to flatten out meaning the accuracy of the clustering does not improve massively as more clusters are added. The elbow point is the point where the distortion stops decreasing rapidly.

The Python library Scikit-learn or sklearn is a very useful package for machine learning. We have used it to generate the toy dataset shown in Figure 2.1. We used the function *sklearn.datasets.make_blobs* which generates a specified number of Gaussian-distributed "blobs". One can specify the required number of samples (n), the number of blobs (K), the coordinates of the centre of each blob ($\boldsymbol{\mu}_k$) and the covariance matrix of each blob ($\boldsymbol{\Sigma}_k$) among other parameters. The names of these parameters respectively are *n_samples, n_features, centers* and *cluster_std*. Figure 2.1 displays an example of the elbow method for a toy dataset. We have chosen parameters $n = 1000, K = 3$, $\boldsymbol{\mu}_1 = (0, 1.5)$, $\boldsymbol{\mu}_2 = (-1.5, -1.5)$, $\boldsymbol{\mu}_3 = (1.5, -1.5)$ and $\boldsymbol{\Sigma}_k = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$ for all three blobs.

Figure 2.1



(a) Visualisation of a simple two-dimensional dataset, Toy 1.

(b) The elbow curve for Toy 1. The elbow point is clearly at K=3.

Clearly the elbow point for this dataset is at K=3 meaning the optimal number of clusters is K=3. This agrees with the visualisation of the dataset on the left as there are clearly three clusters. In practice one cannot always identify an elbow point as it is very sensitive to the dataset being clustered (Ketchen & Shook 1996). Note also that the value of K at the elbow point is not always the optimal value. Other values could be better in practice.

2.3 Advantages and disadvantages

The previous section outlined one major disadvantage of the k-means algorithm: choosing the number of clusters K (Raschka 2015). Another disadvantage of the method is that every observation is clustered. This means that the method is not robust to outliers (Sonagara & Badheka 2014). Even if an observation is completely different to the rest of the dataset, it will still be assigned to a cluster. One way to detect outliers is to look for clusters with very few observations in. These are most likely outliers and so it is a common post-processing step to remove such clusters. Another disadvantage of k-means clustering is that the algorithm performs poorly when the clusters have different sizes and densities (Sonagara & Badheka 2014). Two distinct clusters could be close together with one very large and the other very small. In this case, observations close to the edge of the larger cluster could be closer to the centroid of the smaller cluster and hence be wrongfully assigned to it. Similar issues can occur when clusters have different densities. A very sparse cluster could be partitioned into several clusters if the wrong value of K is chosen. The nature of Euclidean distance is such that the algorithm performs poorly when clusters are not spherical in shape. Clusters do not necessarily have to be spherical in shape. The vast amount of different definitions of exactly what a cluster is is testament to that. The only requirement for observations in a cluster is that they are close to each other and this closeness can be measured in a variety of ways, not just using Euclidean distance. Another issue that arises from k-means clustering is that the method is very sensitive to scaling (Berkhin 2002). For example, one feature could have range [0,1] and another $(-\infty, \infty)$. Clearly there will be a greater weighting towards the second feature. Therefore it is common practice to scale or normalise each feature so that they have zero mean and unit variance.

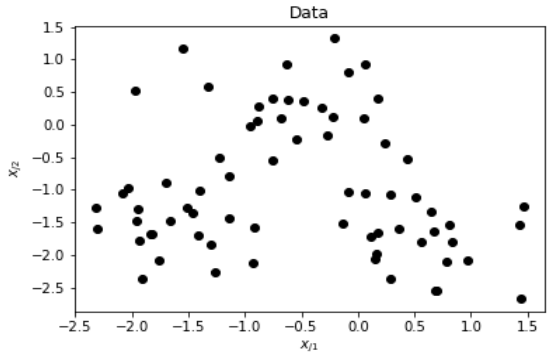
Now onto some of the advantages of the k-means algorithm. Compared to the majority of other clustering methods, k-means is very efficient (Schubert 2017). It has computational complexity $\mathcal{O}(Knpi)$ where i is the number of iterations required. The notation g(x) $= \mathcal{O}(f(x))$ means that $\left|\frac{f(x)}{g(x)}\right|$ is bounded as $x \to \infty$ for a real-valued function f and $x \in \mathcal{R}$. The value of i should be relatively small for datasets which have a natural cluster structure and so in practice the algorithm is very quick. The final centroids can be very useful in extracting information about the cluster members. For example, consider a database containing the customers of an insurance company. One could cluster the dataset based on information such as age, occupation, income, postcode, credit history and past claims. Merely clustering the customers into different groups would not be particularly useful. But looking at the centroids could tell us very important information about the type of customers in each group. One could find a cluster consisting of young, low income people with multiple past claims. Alternatively there could be a cluster containing middle-aged, wealthy customers living in affluent areas. The company could then target the groups with specific offers and advertisements in order to boost sales. One property of the algorithm is that every repetition yields a different result (Santini 2016). This could be seen as a positive or a negative depending on the dataset. We

could find a new cluster that had not previously been found. Alternatively, we could find that too many new clusters are being found with each repetition making it hard to draw meaningful conclusions from the clustering.

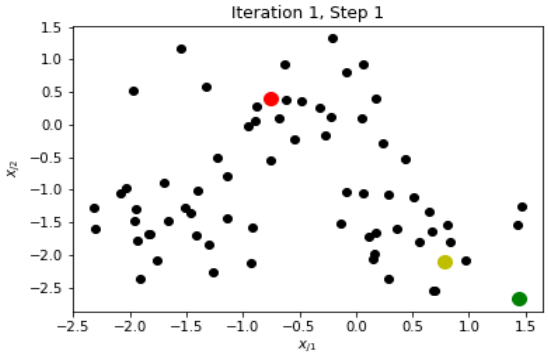
2.4 An example of k-means clustering

We shall now take a look at the k-means algorithm in practice. We will go through step by step and see the outcome of each iteration. We have used the Python function *sklearn.datasets.make_blobs()* from earlier with the parameters $n = 70, K = 3$, $\boldsymbol{\mu}_1 = (-0.7, 0.2)$, $\boldsymbol{\mu}_2 = (-1.5, -1.5)$, $\boldsymbol{\mu}_3 = (0.5, -2)$ and $\boldsymbol{\Sigma}_k = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$ for all three blobs. Figure 2.2b shows step 1, the initial cluster centroid choices. These have been randomly picked from the dataset and are visible in the plot as the larger, coloured dots. Figure 2.2c shows the results of step 3. Each observation has been assigned to the cluster with the closest centroid. Figure 2.2d shows the results of step 4 in which the centroids have been updated as the mean vectors of the observations in each cluster. Figures 2.2e, 2.2f, 2.2g and 2.2h show the results at the end of each iteration. The cluster assignments no longer change after iteration 7 meaning the algorithm has ended. Note that as expected, the distortion of each clustering decreases as the algorithm proceeds. This is because the observations are getting closer and closer to their centroid. Despite the initial choice of centroids appearing to be a poor one, the centroids move into more appropriate positions as the algorithm iterates showing why the k-means algorithm is one of the most widely-used in the field of cluster analysis.

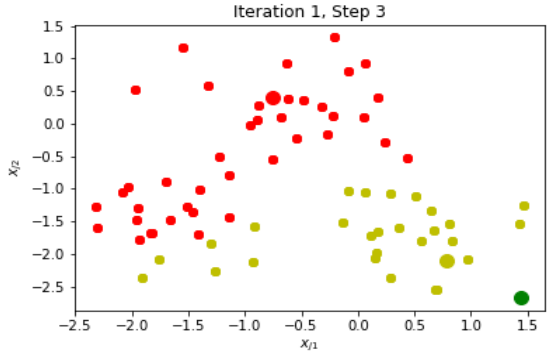
Figure 2.2: K-means clustering of a toy dataset



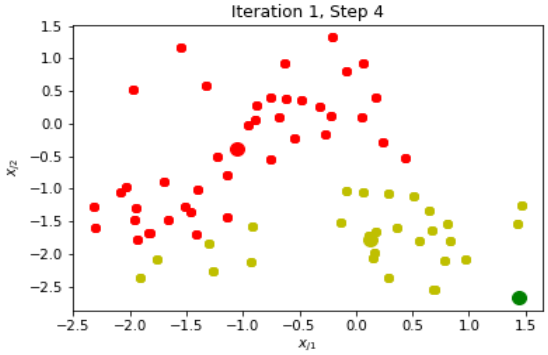
(a) Visualisation of two-dimensional dataset



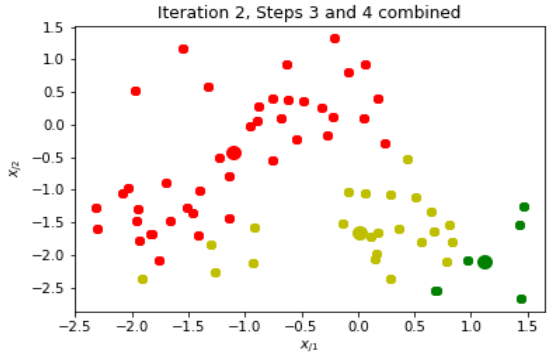
(b) Random cluster centroid initialisation.



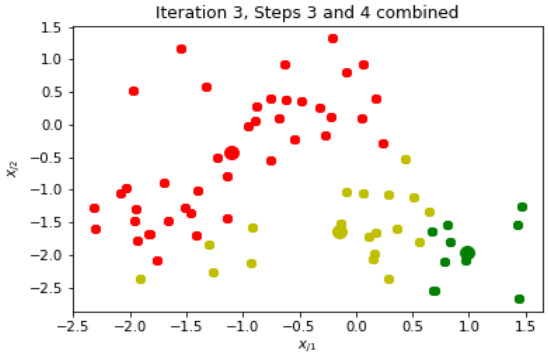
(c) Initial cluster assignment



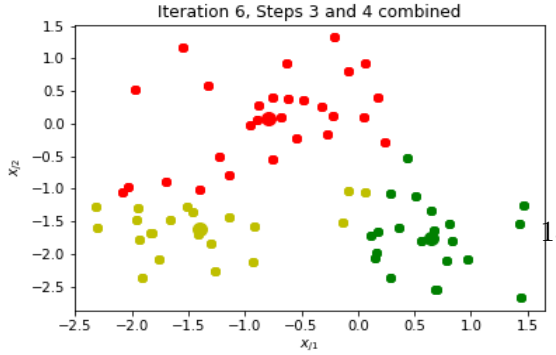
(d) Updated centroids. Distortion = 152



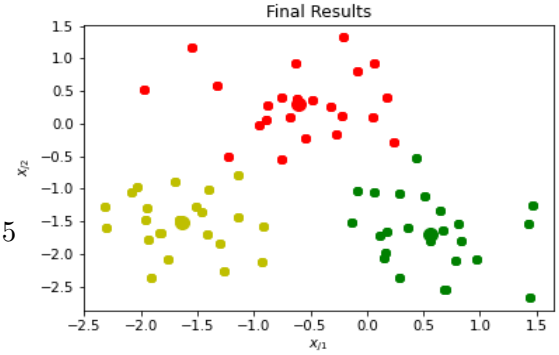
(e) Updated clustering. Distortion = 151



(f) Updated clustering. Distortion = 150



(g) Updated clustering. Distortion = 141



(h) Final clustering. Distortion = 102

Chapter 3

OPTICS

3.1 Background

The nature of clusters within multi-dimensional datasets is such that their intrinsic structure cannot be summarised by global parameters. Therefore it is usually pointless using clustering methods which take as input a global density parameter (Aggarwal & Reddy 2014). One option is to use a density-based method multiple times with a range of different parameter settings, such as DBSCAN (Ester et al. 1996). However, this is very difficult in practice since there is an infinite number of combinations for these parameters. A solution to this problem was proposed in 1999 by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander called OPTICS (Ordering Points To Identify the Clustering Structure). This algorithm produces an ordering of the observations in a dataset with respect to its density-based clustering structure such that observations which are close to each other are neighbours in the ordering.

As stated in Subsection 1.3.2, in density-based clustering, a cluster is defined as an area of high density surrounded by an area of lower density. There are two key parameters which are involved with the majority of density based clustering algorithms, $\epsilon \in \mathbb{R}_{\geq 0}$ and $MinPts \in \mathbb{N}$. Each cluster must contain an observation with at least $MinPts$ observations within an ϵ-neighbourhood (including $\mathbf{x}_j$). We shall now introduce some concepts as defined in (Ankerst et al. 1999).

Definition 3.1.1 (Directly density-reachable)**.** An observation $\mathbf{x}_j$ is *Directly density-reachable* from an observation $\mathbf{x}_l$ with respect to ϵ and $MinPts$ in a dataset $\mathbf{x}$ if

1. $\mathbf{x}_j \in N_\epsilon(\mathbf{x}_l)$ where $N_\epsilon(\mathbf{x}_l) = \{\mathbf{x}_i \in \mathbf{x} : d(\mathbf{x}_l, \mathbf{x}_i) \leq \epsilon\}$ for some distance function d.
2. $|N_\epsilon(\mathbf{x}_l)| \geq MinPts$ where $|N_\epsilon(\mathbf{x}_l)|$ denotes the cardinality of the set $N_\epsilon(\mathbf{x}_l)$.

As we stated earlier, if condition 2 holds for an observation $\mathbf{x}_l$, then $\mathbf{x}_l$ is a core point.

Definition 3.1.2 (Density-reachable)**.** An observation $\mathbf{x}_j$ is *density-reachable* from an observation $\mathbf{x}_l$ with respect to ϵ and $MinPts$ in a dataset $\mathbf{x}$ if there exists a sequence

of observations, $\mathbf{x}_1$,...,$\mathbf{x}_n$ such that $\mathbf{x}_i \in \mathbf{x}$, $\mathbf{x}_1 = \mathbf{x}_l$, $\mathbf{x}_n = \mathbf{x}_j$ and $\mathbf{x}_{i+1}$ is directly density-reachable from $\mathbf{x}_i$ with respect to ϵ and $MinPts$.

Note that this is not a symmetric property, i.e. $\mathbf{x}_j$ being density-reachable from $\mathbf{x}_l$ does not guarantee $\mathbf{x}_l$ being density reachable from $\mathbf{x}_j$. Only core points are mutually density-reachable.

Definition 3.1.3 (Density-connected)**.** An observation $\mathbf{x}_j$ is *density-connected* to an observation $\mathbf{x}_l$ with respect to ϵ and $MinPts$ in a dataset $\mathbf{x}$ if there exists another observation $\mathbf{x}_i \in \mathbf{x}$ such that both $\mathbf{x}_j$ and $\mathbf{x}_l$ are density-reachable from $\mathbf{x}_i$ with respect to ϵ and $MinPts$.

Note that density-connectivity is a symmetric relation. Figure 3.1 depicts a two-dimensional toy dataset along with some of the above definitions.

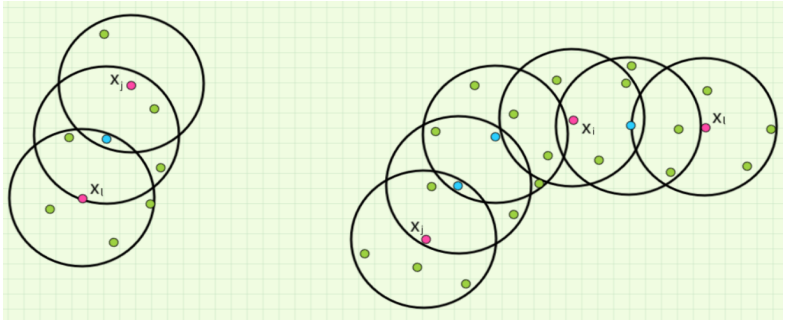


Figure 3.1: ($MinPts = 6$, circles are of radius ϵ) On the left: $\mathbf{x}_j$ is density-reachable from $\mathbf{x}_l$ but $\mathbf{x}_l$ is not density-reachable from $\mathbf{x}_j$. On the right: $\mathbf{x}_j$ and $\mathbf{x}_l$ are density-connected to each other by $\mathbf{x}_i$.

In this chapter it is more convenient to let C_k denote the set containing the observations themselves rather than the indices of the observations in a cluster.

Definition 3.1.4 (Clusters)**.** A *cluster* C_k is a non-empty subset of $\mathbf{x}$ with respect to ϵ and $MinPts$ which satisfies the following conditions:

1. $\forall\ \mathbf{x}_j, \mathbf{x}_l \in \mathbf{x}$, if $\mathbf{x}_l \in C_k$ and $\mathbf{x}_j$ is density-reachable from $\mathbf{x}_l$ with respect to ϵ and $MinPts$, then $\mathbf{x}_j \in C_k$. (Maximality)

2. $\forall\ \mathbf{x}_j, \mathbf{x}_l \in C_k$, $\mathbf{x}_j$ is density-connected to $\mathbf{x}_l$ with respect to ϵ and $MinPts$. (Connectivity)

Note that any observation that is not in any cluster is an outlier/noise. Here is a reminder of the types of points that were mentioned in the introduction.

$$\text{Type}(\mathbf{x}_j) = \begin{cases} \text{Core}, & \text{if } |N_\epsilon(\mathbf{x}_j)| \geq MinPts \\ \text{Border}, & \text{if } |N_\epsilon(\mathbf{x}_j)| < MinPts \cap \exists\ \mathbf{x}_l : \mathbf{x}_j \text{ directly density-reachable from } \mathbf{x}_l \\ \text{Noise}, & \text{if } |N_\epsilon(\mathbf{x}_j)| < MinPts \cap \mathbf{x}_j \text{ not directly density-reachable from } \mathbf{x}_l\ \forall\ \mathbf{x}_l \in \mathbf{x} \end{cases} \tag{3.1}$$

A density-based cluster with respect to ϵ and $MinPts$ will always be contained within a density-connected cluster with respect to ϵ' and $MinPts$ if the first cluster has higher density, (i.e. $\epsilon < \epsilon$'). Therefore it is possible to develop the DBSCAN algorithm described in the introduction to produce clusterings of a dataset using different values of ϵ simultaneously. However, for the results to be reproducible there would have to be an order in which observations are clustered. It is necessary at all times to cluster the observation which is density-reachable with respect to the smallest ϵ value possible. This guarantees that the clusters with the highest density are always clustered first. The OPTICS algorithm works in a similar way by performing clustering for an infinite number of ϵ_i values below a given generating distance ϵ. Instead of performing the clustering itself, it stores the order in which each observation should be processed and two key values for each observation which can be used by a density-based clustering like DBSCAN. These key values are the *core-distance* and *reachability-distance* which are defined below (Ankerst et al. 1999).

Definition 3.1.5 (Core-distance)**.** For an observation $\mathbf{x}_j \in \mathbf{x}$ and the parameters ϵ and $MinPts$, the *core-distance* of $\mathbf{x}_j$ is defined as

$$\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_j) = \begin{cases} d_{MinPts}(\mathbf{x}_j), & \text{if } |N_\epsilon(\mathbf{x}_j)| \geq MinPts \\ \infty, & \text{otherwise} \end{cases} \tag{3.2}$$

where as before $|N_\epsilon(\mathbf{x}_j)|$ is the cardinality of the set of points in the ϵ-neighbourhood of $\mathbf{x}_j$ and $d_{MinPts}(\mathbf{x}_j)$ is the distance between $\mathbf{x}_j$ and its $MinPts^{th}$ closest neighbour. Any distance function can be used. In other words the core-distance of an observation $\mathbf{x}_j$ is the smallest distance ϵ' between $\mathbf{x}_j$ and another point $\mathbf{x}_l$ in its ϵ-neighbourhood such that $\mathbf{x}_j$ is a core point with respect to ϵ' and $MinPts$ if $\mathbf{x}_l \in N_\epsilon(\mathbf{x}_j)$.

Definition 3.1.6 (Reachability-distance)**.** For two observations $\mathbf{x}_j, \mathbf{x}_i \in \mathbf{x}$ and the parameters ϵ and $MinPts$, the *reachability-distance* of $\mathbf{x}_j$ with respect to $\mathbf{x}_i$ is defined as

$$\textit{reach-dist}_{\epsilon,MinPts}(\mathbf{x}_j, \mathbf{x}_i) = \begin{cases} \max(\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_i), d(\mathbf{x}_i, \mathbf{x}_j)), & \text{if } |N_\epsilon(\mathbf{x}_i)| \geq MinPts \\ \infty, & \text{otherwise} \end{cases} \tag{3.3}$$

In other words, if $\mathbf{x}_i$ is a core point, the reachability-distance of $\mathbf{x}_j$ with respect to $\mathbf{x}_i$ is the smallest distance such that $\mathbf{x}_j$ is directly density-reachable from $\mathbf{x}_i$. If $\mathbf{x}_i$ is not a core point then the reachability-distance is undefined. Figure 3.2 illustrates the concepts of reachability-distance and core-distance.

3.2 OPTICS algorithm

The OPTICS algorithm produces an ordering of the dataset, storing the core-distance and a reachability-distance for each observation. This allows one to find any density-based cluster in the dataset with respect to $MinPts$ and a ϵ' value so long as it is smaller

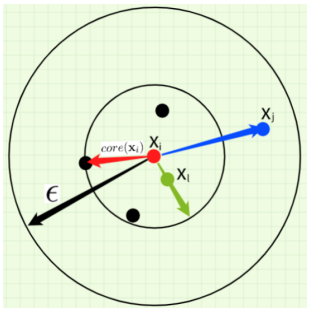


Figure 3.2: ($MinPts = 5$). Blue arrow shows $reach\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_j, \mathbf{x}_i)$ and green arrow shows $reach\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_l, \mathbf{x}_i) = core\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_i)$ which is the distance represented by the red arrow.

than the generating distance ϵ. The OPTICS algorithm takes as inputs a dataset $\mathbf{x}$, a generating distance ϵ and $MinPts$ and its steps are shown in Algorithms 2 and 3 (Ankerst et al. 1999).

Since UpdateQueue($\mathbf{x}_j$) updates the queue in order of increasing $r_{\epsilon,MinPts}()$ and due to the fact that once an observation is processed, its reachability-distance in O is fixed it follows that the final reachability-distance for an observation $\mathbf{x}_j$ in O can be written as $r_{\epsilon,MinPts}(\mathbf{x}_j) = \min(reach\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_j, \mathbf{x}_i) \mid \mathbf{x}_i \text{ comes before } \mathbf{x}_j \text{ in } O \cap d(\mathbf{x}_j, \mathbf{x}_i) \leq \epsilon)$.

3.3 Extracting the clustering following OPTICS

Once we have executed the OPTICS algorithm we can extract any density-based clustering from the ordering produced. To do this we can scan the ordering and assign a cluster to each observation based on their core-distance and reachability-distance with respect to $MinPts$ and a clustering distance $\epsilon' < \epsilon$. Algorithm 4, ExtractClustering(O, $MinPts$,ϵ') is used to extract the clusters from the ordering generated by OPTICS (Ankerst et al. 1999). It takes as parameters the ordering of $\mathbf{x}$ produced by OPTICS: O, $MinPts$ and ϵ' which must be smaller than the generating distance ϵ used in OPTICS. We check if $r_{\epsilon,MinPts}(\mathbf{x}_j)$ is greater than ϵ' because in this case $\mathbf{x}_j$ is not density-reachable from any of the previous observations in O with respect to $MinPts$ and ϵ'. This is because if it was density-reachable from a previous observation in O, then its assigned reachability-distance would have been less than ϵ'. If it is the case that $r_{\epsilon,MinPts}(\mathbf{x}_j) > \epsilon'$ then we can look at $core\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_j)$ as well. If $core\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_j) \leq \epsilon'$, then $\mathbf{x}_j$ is a core point with respect to $MinPts$ and ϵ' and so we can begin a new cluster at this point. If however, $core\text{-}dist_{\epsilon,MinPts}(\mathbf{x}_j) > \epsilon'$, then $\mathbf{x}_j$ cannot be part of a cluster and so we label it as an outlier/noise. On the other hand, if $r_{\epsilon,MinPts}(\mathbf{x}_j)$ is less than or equal to ϵ', then $\mathbf{x}_j$ is density-reachable from a preceding observation in the ordering with respect to $MinPts$ and ϵ' and so is added to its cluster.

Algorithm 2: OPTICS algorithm

Input: A dataset $\mathbf{x}$, a generating distance ϵ and $MinPts$.

Output: An ordering of $\mathbf{x}$ containing a core-distance, $\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_j)$ and a reachability-distance, $r_{\epsilon,MinPts}(\mathbf{x}_j)$ for each $\mathbf{x}_j \in \mathbf{x}$.

Begin with empty queue, Q and order list, O
for *each* $\boldsymbol{x}_j \in \boldsymbol{x}$ **do**
  if $\boldsymbol{x}_j$ *is unprocessed* **then**
    Find the ϵ-neighbourhood of $\mathbf{x}_j$, $N_\epsilon(\mathbf{x}_j)$.
    Mark $\mathbf{x}_j$ as processed.
    Set the reachability-distance of $\mathbf{x}_j$, $r_{\epsilon,MinPts}(\mathbf{x}_j)$ to ∞.
    Calculate $\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_j)$.
    Put $\mathbf{x}_j$ in O along with $\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_j)$ and $r_{\epsilon,MinPts}(\mathbf{x}_j)$.
    if $\textit{core-dist}_{\epsilon,MinPts}(\boldsymbol{x}_j) \neq \infty$ **then**
      Execute UpdateQueue($\mathbf{x}_j$,$N_\epsilon(\mathbf{x}_j)$,ϵ,$MinPts$).
      while Q *is not empty* **do**
        Select the next unprocessed $\mathbf{x}_i$ in Q.
        Find the ϵ-neighbourhood of $\mathbf{x}_i$, $N_\epsilon(\mathbf{x}_i)$.
        Mark $\mathbf{x}_i$ as processed.
        Calculate $\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_i)$.
        Put $\mathbf{x}_i$ in O along with $\textit{core-dist}_{\epsilon,MinPts}(\mathbf{x}_i)$ and $r_{\epsilon,MinPts}(\mathbf{x}_i)$.
        if $\textit{core-dist}_{\epsilon,MinPts}(\boldsymbol{x}_i) \neq \infty$. **then**
          Execute UpdateQueue($\mathbf{x}_i$,$N_\epsilon(\mathbf{x}_i)$,ϵ,$MinPts$).
return O

Algorithm 3: UpdateQueue algorithm

Input: An observation $\mathbf{x}_j$, its ϵ-neighbourhood $N_\epsilon(\mathbf{x}_j)$, a generating distance ϵ and $MinPts$.

Output: The queue for processing $\mathbf{x}$, Q.

for *each unprocessed* $\boldsymbol{x}_l \in N_\epsilon(\boldsymbol{x}_j)$ **do**
  Calculate $\textit{reach-dist}_{\epsilon,MinPts}(\mathbf{x}_l, \mathbf{x}_j)$.
  if $\boldsymbol{x}_l$ *is not already in* Q **then**
    Set $r_{\epsilon,MinPts}(\mathbf{x}_l)$ equal to $\textit{reach-dist}_{\epsilon,MinPts}(\mathbf{x}_l, \mathbf{x}_j)$ and insert it in Q in order of increasing $r_{\epsilon,MinPts}()$.
  else if $\boldsymbol{x}_l$ *is already in* Q *but* $\textit{reach-dist}_{\epsilon,MinPts}(\boldsymbol{x}_l, \boldsymbol{x}_j) < r_{\epsilon,MinPts}(\boldsymbol{x}_l)$ **then**
    Set $r_{\epsilon,MinPts}(\mathbf{x}_l)$ equal to $\textit{reach-dist}_{\epsilon,MinPts}(\mathbf{x}_l, \mathbf{x}_j)$ and move $\mathbf{x}_l$ higher in Q based on its new value for $r_{\epsilon,MinPts}(\mathbf{x}_l)$.
return Q

Algorithm 4: ExtractClustering algorithm

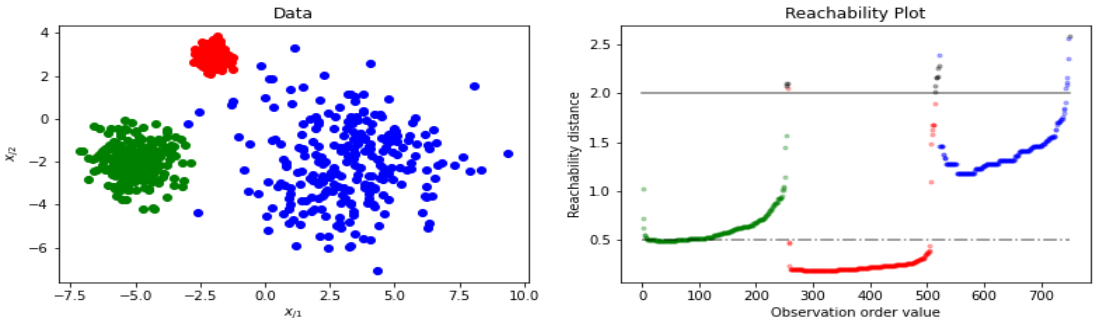
Input: A cluster ordering O,ϵ', $MinPts$.
Output: A clustering.

```
for each x_j ∈ O do
    if r_{ε,MinPts}(x_j) > ε' then
        if core-dist_{ε,MinPts}(x_j) ≤ ε' then
            assign x_j to the next empty cluster.
        else
            label x_j as noise.
    else
        assign x_j to the current cluster.
return Clustering
```

3.4 Reachability plots

Once we have the results of the OPTICS algorithm, it is common to visualise the results graphically using a *reachability plot*. This is a plot of $r_{\epsilon,MinPts}(\mathbf{x}_j)$ for every $\mathbf{x}_j$ in O, with the $\mathbf{x}_j$'s in the same order as they are in O. The reachability plot of a dataset is on the whole, quite insensitive to the choice of parameters; $MinPts$ and ϵ. This is because there are a range of values for these parameters such that the clustering-structure of the dataset is always easily visible. As long as the parameters are high enough, the results will usually be the same. Figure 3.3 shows an example of a reachability plot for a toy two-dimensional dataset which we have generated by hand using the Python function *numpy.random.randn* which randomly generates a specified number of points from the standard normal distribution of a specified number of dimensions. By multiplying these points by a constant and adding them to a different constant we can create clusters. The dataset contains 750 observations, each with 2 features ($n = 750$ and $p = 2$). The observations form three clusters ($K = 3$).
The clusters are are represented by the dips in the reachability-distance. This makes sense because in a cluster, the observations are more densely packed together and so each observation will have small reachability-distance from the preceding observation in the ordering. The red cluster clearly has the highest density which agrees with the reachability plot since the red dip has the lowest average reachability-distance of the dips. The reachability plot has two grey horizontal lines at $\epsilon = 2$ and $\epsilon = 0.5$. Since all three clusters lie underneath the line at $\epsilon = 2$, this means that running algorithm ExtractClustering(O, $MinPts = 50$ and $\epsilon' = 2$) - where O is the ordering produced by OPTICS on the toy-dataset - should extract the clusters reasonably well. However, running the algorithm with $\epsilon' = 0.5$ would not produce the same clustering because at this point the reachability-distance of many of the observations in the blue and green clusters would be greater than ϵ' and so would either be assigned to a new cluster or labelled as noise.

Figure 3.3: Reachability plot of a toy dataset

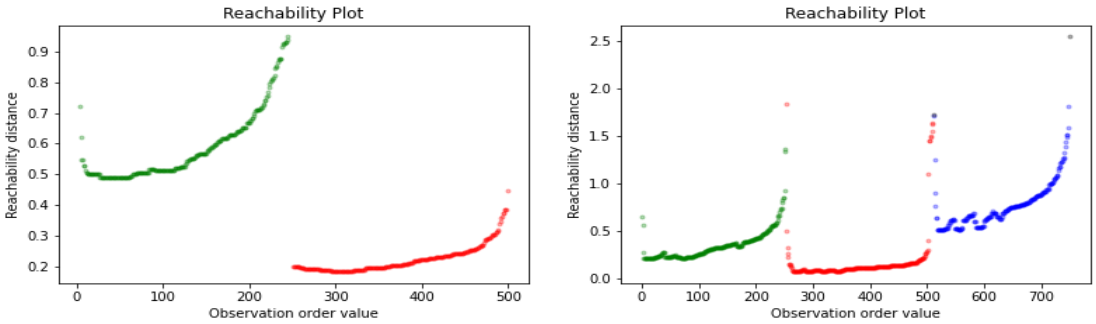


(a) Visualisation of two-dimensional toy dataset which we are running OPTICS on.

(b) Reachability plot of the dataset to the left. Ordering generated by OPTICS with parameters $\epsilon = \infty$ and $MinPts = 50$

We will now look at the effect that changing the OPTICS parameters ϵ and $MinPts$ has on the results produced. Figure 3.4 shows the reachability plots for the dataset in Figure 3.3a with the OPTICS parameters slightly changed.

Figure 3.4: Reachability plot of a toy dataset



(a) Reachability plot of ordering produced by OPTICS with $\epsilon = 1$ and $MinPts = 50$

(b) Reachability plot of ordering produced by OPTICS with $\epsilon = \infty$ and $MinPts = 14$

Clearly, decreasing the value of ϵ decreases the number of clusters visible in the reachability plot. This is because the number of observations with undefined reachability-distance increases and so their reachability-distance is greater than ϵ meaning the lower density clusters cannot be formed. The reachability plot does not show such observations as it is impossible to plot an observation with undefined reachability-distance. The reachability plot on the right shows the effect of decreasing $MinPts$. Whilst the overall shape of the curve is similar, the curve itself is much more jagged in nature. This happens because fewer points have reachability-distance equal to their core-distance.

3.5 ξ method for hierarchical clusters

When our dataset has a cluster structure in which no cluster is contained within another cluster, the ExtractClustering() method works well. However, it is much less effective when the clusters have a hierarchical structure, i.e. the clusters are nested. Luckily there exists another method of extracting the clusters when this is the case. Before we discuss it, we must first introduce a few more concepts.

For the next section, it is more convenient to think of the ordering produced by OPTICS, as a mapping from the integer interval {1,...,n} to the dataset $\mathbf{x}$, i.e. $O : \{1, ..., n\} \to \mathbf{x}$. So the first observation in the ordering can be written $O(1)$. The ordering has the following properties:

- The first observation in the ordering can be any of the observations in the dataset.
- The next observation in the dataset is always the observation with the least reachability distance from the preceding observations out of all of the unprocessed observations.

As mentioned earlier, a cluster is represented by a dent in the reachability plot. Shown below in Figure 3.5 is a toy reachability plot. The reachability plot has one dent in the curve meaning the ordering it is based on contains one cluster. This cluster begins at observation number 3 and ends at observation number 11. This is because the third observation has a high reachability-distance meaning it is far away from the first two observations but observation 4 has comparatively low reachability-distance. This means that observation 4 is relatively close to one of observations 1,2 or 3. It must be closest to observation 3 because otherwise it would have index 3 in the ordering rather than 4. Observation 11 is the last observation in the cluster because it has low reachability-distance indicating it is close to its predecessors (i.e. the other points in the cluster) whilst observation 12 has high reachability-distance meaning it is not close to the cluster.

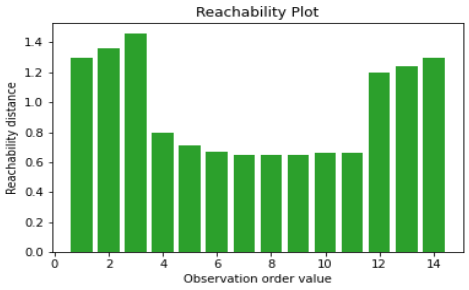


Figure 3.5: A toy reachability plot.

In reality, it is not always so easy to determine the start and end of a cluster in a reachability plot. The sudden increases and decreases in reachability-distance in Figure

3.5 may not be so sudden in the reachability plots of real world datasets. It is for this reason that another method of identifying clusters exists. Before describing this method, we must first introduce a few more definitions from (Ankerst et al. 1999).

Definition 3.5.1 (ξ-steep points)**.** An ordered observation $O(j) \in \mathbf{x}\backslash\{O(n)\}$ is a *ξ-steep upward point* if and only if it is ξ% lower than its successor, $O(j+1)$. i.e.

$$UpPoint_\xi(O(j)) \Leftrightarrow r_{\epsilon,MinPts}(O(j)) \leq r_{\epsilon,MinPts}(O(j+1)) \times (1-\xi) \tag{3.4}$$

An ordered observation $O(j) \in \mathbf{x}\backslash\{O(n)\}$ is a *ξ-steep downward point* if and only if its successor, $O(j+1)$ is ξ% lower than $O(j)$. i.e.

$$DownPoint_\xi(O(j)) \Leftrightarrow r_{\epsilon,MinPts}(O(j)) \times (1-\xi) \geq r_{\epsilon,MinPts}(O(j+1)) \tag{3.5}$$

The majority of clusters start with ξ-steep points before a series of points where the reachability-distance is relatively even before ending with another series of ξ-steep points. More formally these are known as ξ-steep areas and can be defined as follows:

Definition 3.5.2 (ξ-steep areas)**.** A set of observations, $O(I)$ where I is an integer interval $I = \{a, ..., b\} \subset \{1, ..., n\}$ is called a *ξ-steep upward area*, $UpArea_\xi(O(I))$ if and only if the following conditions are satisfied:

1. $O(a)$ is a ξ-steep upward point, i.e. $UpPoint_\xi(O(a))$
2. $O(b)$ is a ξ-steep upward point i.e. $UpPoint_\xi(O(b))$
3. Each point between $O(a)$ and $O(b)$ has reachability-distance at least as high as the observation before it. i.e. $\forall\ O(j+1)$ s.t. $O(j+1) \in O(I)$, $r_{\epsilon,MinPts}(O(j)) \leq r_{\epsilon,MinPts}(O(j+1))$
4. There is not more than $MinPts$ consecutive points that are not ξ-steep upward points, i.e. $\forall$ integer intervals $I' = \{a', .., b'\} \subset \{a, ..., b\}$ *s.t.* $O(I') \subset O(I)$, let $A = \{O(j) \in O(I') : O(j)$ is not ξ-steep upward point$\}$. Then $|A| \leq MinPts$, where $|A|$ denotes the cardinality of the set A.
5. $O(I)$ is maximal, i.e. $\forall\ J \subset \{1, ..., n\}$ *s.t.* $O(I) \subset O(J)$, $UpArea_\xi(O(J)) \Rightarrow I = J$.

A ξ-steep downward area, $DownArea_\xi(O(I))$ is defined analogously.

Definition 3.5.3 (ξ-cluster)**.** A set of observations, $O(C)$ where C is an integer interval $C = \{a, ..., b\} \subset \{1, ..., n\}$ is called a *ξ-cluster* if and only if it satisfies the following conditions: $Cluster_\xi(O(C)) \Leftrightarrow \exists\ D = \{a_D, ..., b_D\},\ U = \{a_U, ..., b_U\}, D, U \subset C$ with:

1. $DownArea_\xi(O(D))$ and $a \in D$
2. $UpArea_\xi(O(U))$ and $b \in U$
3. (a) $b + 1 - a \geq MinPts$

(b) $\forall j : a_D < j < b_U$,
$r_{\epsilon,MinPts}(O(j)) \leq \min(r_{\epsilon,MinPts}(O(a_D)), r_{\epsilon,MinPts}(O(b_U))) \times (1 - \xi)$

4. $(a, b) = \begin{cases} (\max\{j \in D | r(O(j)) > r(O(b_U + 1))\}, b_U), & \text{if } r(O(a_D)) \times (1 - \xi) \geq r(O(b_U + 1)) \\ (a_D, \min\{j \in U | r(O(j)) > r(O(a_D))\}), & \text{if } r(O(b_U + 1)) \times (1 - \xi) \geq r(O(a_D)) \\ (a_D, b_U), & \text{otherwise} \end{cases}$

where $r(O(j))$ denotes $r_{\epsilon,MinPts}(O(j))$ for space-saving reasons.

The first two conditions merely state that the start of the cluster must be contained in a ξ-steep downward area and the end of the cluster must be contained in a ξ-steep upward area. Condition 3(a) states that the cluster must contain at least $MinPts$ observations while 3(b) states that the reachability-distance of every point between the start and end of the cluster must be at least $\xi\%$ lower than both $O(a_D)$ and $O(b_U)$. Condition 4 decides the start and end points of the cluster. There are three cases depending on the reachability-distance of $O(a_D)$ and $O(b_U + 1)$. If the two reachability-distances are at most $\xi\%$ apart, then the cluster starts at $O(a_D)$ and ends at $O(b_U)$. Second, if the reachability-distance of $O(a_D)$ is more than $\xi\%$ higher than the reachability-distance of $O(b_U + 1)$, then the cluster ends at $O(b_U)$, but starts at the point in D, that has approximately the same reachability value as $O(b_U + 1)$. Otherwise if the reachability-distance of $O(b_U + 1)$ is more than $\xi\%$ higher than $O(a_D)$, the cluster starts at the first point of $O(a_D)$ and ends at the point in U, that has approximately the same reachability value as $O(a_D)$.

There is a simple algorithm (Algorithm 5) which can identify all ξ-clusters in a dataset by going through the cluster ordering and finding all of the ξ-steep upward and downward areas, merging them into clusters if the conditions in Definition 3.5.3 are satisfied (Ankerst et al. 1999).

Note that s_{start} denotes the index of the start point in s and D_{end} denotes the index of the end point in D. C_{end} and U_{end} are defined similarly. Note also that we have introduced a variable m which stands for maximum-in-between value. This allows us to streamline the algorithm by changing condition 3(b) in Definition 3.5.3 slightly. Condition 3(b): $\forall j : a_D < j < b_U$,
$r_{\epsilon,MinPts}(O(j)) \leq \min(r_{\epsilon,MinPts}(O(a_D)), r_{\epsilon,MinPts}(O(b_U))) \times (1 - \xi)$ can be written equivalently as

1. $\forall j : a_D < j < b_U, r_{\epsilon,MinPts}(O(j)) \leq r_{\epsilon,MinPts}(O(a_D)) \times (1 - \xi) \cap$
2. $\forall j : a_D < j < b_U, r_{\epsilon,MinPts}(O(j)) \leq r_{\epsilon,MinPts}(O(b_U)) \times (1 - \xi)$

These statements are equivalent to:

1. $\max\{j \mid a_D < j < b_U, r_{\epsilon,MinPts}(O(j))\} \leq r_{\epsilon,MinPts}(O(a_D)) \times (1 - \xi) \cap$
2. $\max\{j \mid a_D < j < b_U, r_{\epsilon,MinPts}(O(j))\} \leq r_{\epsilon,MinPts}(O(b_U)) \times (1 - \xi)$

Algorithm 5: ξ-ExtractClustering

Input: A cluster ordering O and ξ.
Output: A clustering of $\mathbf{x}$.

Begin with empty set of ξ-steep downward areas, S, empty set of clusters C_{set}, let $i = 1$, $m = 0$
while $i < n$ **do**
  let $m = \max(m, r_{\epsilon,MinPts}(O(i)))$
  if *O(i) is the start of a ξ-steep downward area, D* **then**
    if $m \neq \infty$ **then**
      for *each* $s \in S$ **do**
        if $m > r_{\epsilon,MinPts}(O(s_{start})) \times (1 - \xi)$ **then**
          remove s from S
      for *each* $s \in S$ **do**
        $m_s = max(m_s, m)$
    set $m_D = 0$
    add D to S
    set i = $D_{end} + 1$
    set m = $r_{\epsilon,MinPts}(O(i))$
  else if *O(i) is the start of a ξ-steep upward area, U* **then**
    if $m \neq \infty$ **then**
      for *each* $s \in S$ **do**
        if $m > r_{\epsilon,MinPts}(O(s_{start})) \times (1 - \xi)$ **then**
          remove s from S
      for *each* $s \in S$ **do**
        $m_s = max(m_s, m)$
    set i = $U_{end} + 1$
    set m = $r_{\epsilon,MinPts}(O(i))$
    for *each* $s \in S$ **do**
      $C_{start} = s_{start}$
      $C_{end} = U_{end}$
      if $r_{\epsilon,MinPts}(C_{end} + 1) \times (1 - \xi) < m_s$ **then**
        if *D and U satisfy conditions 1,2,3(a)* **then**
          determine the value of (a,b)
          add C to C_{set}
  else
    i = i + 1
return C_{set}

In algorithm 5 we have a global maximum-in-between value, m which keeps track of the maximum reachability-distance between the end of the last ξ-steep (up or down) region found and the current index, i. There is also a maximum-in-between value, m_s for each ξ-steep downward area, s. This contains the maximum reachability-distance between the end of the ξ-steep downward area, s_{end} and the current index, i. The use of the maximum-in-between values cuts out a lot of the computations that would otherwise be needed were we to check condition 3(b) for each potential cluster.

3.6 Advantages and disadvantages

As mentioned earlier, the main advantage of the OPTICS algorithm combined with one of the two cluster extraction algorithms is that it can detect clusters of differing density within the same dataset (Chauhan et al. 2014). In the real world, clusters are almost always guaranteed to be of different densities and so OPTICS would certainly be preferable in the majority of real-world datasets. When combined with the ξ-method for cluster extraction, the OPTICS algorithm performs much better when dealing with hierarchical clusters, i.e. clusters that are nested within other clusters (Kazi & Kurian 2014). These will show up on a reachability plot as a large dent with smaller dents contained within it. When combined with the ExtractClustering algorithm, OPTICS can identify points that it thinks are outliers/noise (Kazi & Kurian 2014). Again this is very useful in the real world since datasets will always contain some outliers be it because of human error, recording errors or unusual sampling methods. It is useful to be able to highlight them so they can be removed or re-recorded. Another advantage of using this combination of ExtractClustering with OPTICS is that arbitrarily shaped clusters can be found rather than simply spherical ones (Kazi & Kurian 2014). In reality, clusters can be shaped in a vast number of ways so its important to be able to detect as many of them as possible. There is also the added benefit that one does not have to specify the number of clusters beforehand which is very useful when dealing with unknown datasets. The computational complexity of OPTICS is $\mathcal{O}(n^2)$ (Ankerst et al. 1999) making it a very efficient algorithm which can perform clusterings on huge datasets in a matter of seconds.

Now onto the disadvantages of OPTICS. One of these is that it requires two parameters, ϵ and $MinPts$ as inputs. Whilst the value for ϵ is not crucial in that it only needs to be large enough to detect the clusters of all densities, the value of $MinPts$ is very important. It determines the number of points required to be in the neighbourhood of an observation for it to be considered a core point. If it is too low, then few points will be classed as such and the reachability plot will be very jagged in shape. Another disadvantage is that, like the majority of clustering methods OPTICS performs poorly on high-dimensional datasets (Kazi & Kurian 2014). This is because as the dimensionality increases, so does the volume of the space, hence observations become sparser and hence there is not as much of a difference in density between clusters and the rest of the dataset. There is also the issue that OPTICS is not entirely deterministic because border points can be density-reachable from several clusters hence assigned to any of them.

Chapter 4

Gaussian mixture models

4.1 Background

Sometimes, it can be useful to assume that a dataset contains observations which are distributed as a mixture of multivariate probability distributions. Each distribution in the mixture can then be considered as a cluster. The most common probability distribution used in such a method is the Gaussian distribution (Aggarwal & Reddy 2014). The Gaussian distribution, otherwise known as the normal distribution is a very common continuous probability distribution for real-valued random variables.

Definition 4.1.1 (Multivariate Gaussian pdf)**.** The p-dimensional multivariate Gaussian distribution has probability density function

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{p/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})) \tag{4.1}$$

where $\boldsymbol{\mu}$ is the p-dimensional mean vector, $\boldsymbol{\Sigma}$ is the $p \times p$ covariance matrix and $|\boldsymbol{\Sigma}|$ denotes its determinant.

In a Gaussian mixture model, each cluster which we will refer to as a component is represented by the parameters of a p-dimensional multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Definition 4.1.2 (Gaussian mixture model pdf)**.** The pdf of the Gaussian mixture model for an observation $\mathbf{x}_j$ can then be written as the linear combination of the individual components:

$$p(\mathbf{x}_j, \Theta) = p(\mathbf{x}_j \mid \pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.2}$$

where Θ is the overall parameter of the mixture model $\{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ and π_k are the component weights otherwise known as mixing probabilities satisfying $0 \leq \pi_k \leq 1$ for $k \in \{1, ..., K\}$ and $\sum_{k=1}^K \pi_k = 1$.

We now introduce a latent variable z_j which is a categorical random variable which takes the values $1, ..., K$ such that $p(z_j = k) = \pi_k$ (Aggarwal & Reddy 2014). This can also be written as $p(z_{jk} = 1) = \pi_k$. Therefore z_j represents the cluster label of observation $\mathbf{x}_j$. Assuming that the distribution of $\mathbf{x}_j$ given $z_j = k$ is given by $p(\mathbf{x}_j|\theta_k)$, the marginal distribution of $\mathbf{x}_j$ can be found by taking the sum of the joint distribution over all values of z_j like in (4.2). Instead of using this single categorical variable z_j to represent the cluster label of $\mathbf{x}_j$ it is more convenient to use a K-dimensional binary random vector $\mathbf{z}_j$. This is a vector containing $K-1$ zeros and 1 one at $z_{jk} = (\mathbf{z}_j)_k$ where $\mathbf{x}_j$ belongs to component k. Note that $\mathbf{z}_j$ satisfies $\sum_{k=1}^{K} z_{jk} = 1$. The marginal distribution of z_j can then be written in terms of the mixing probabilities as

$$p(\mathbf{z}_j) = \pi_1^{z_{j1}} ... \pi_K^{z_{jK}} = \prod_{k=1}^{K} \pi_k^{z_{jk}} \tag{4.3}$$

This means that conditional distribution of $\mathbf{x}_j$ given $\mathbf{z}_j$ can then be written as

$$p(\mathbf{x}_j|\mathbf{z}_j) = \prod_{k=1}^{K} p(\mathbf{x}_j|\theta_k)^{z_{jk}} \tag{4.4}$$

The joint distribution is then given by $p(\mathbf{z}_j)p(\mathbf{x}_j|\mathbf{z}_j)$ and the marginal distribution of $\mathbf{x}_j$ is then

$$p(\mathbf{x}_j) = \sum_{\mathbf{z}_j} p(\mathbf{z}_j)p(\mathbf{x}_j|\mathbf{z}_j) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.5}$$

Clearly this the same as (4.2). Bayes' theorem gives the conditional probability of $z_{jk} = 1$ given $\mathbf{x}_j$:

$$p(z_{jk} = 1|\mathbf{x}_j) = \frac{p(z_{jk} = 1)p(\mathbf{x}_j|z_{jk} = 1)}{\sum_{i=1}^{K} p(z_{ji} = 1)p(\mathbf{x}_j|z_{ji} = 1)} = \frac{\pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \tag{4.6}$$

Here the mixing probability π_k is the prior probability that observation $\mathbf{x}_j$ was generated from component k whilst $p(z_{jk} = 1|\mathbf{x}_j)$ is the posterior probability that observation $\mathbf{x}_j$ came from component k. **From now on we will denote** $p(z_{jk} = 1|\mathbf{x}_j)$ **by** $\gamma(z_{jk})$. Assuming all observations are independently distributed and sampled from the Gaussian distribution, one can write the probability of generating the entire dataset as

$$p(\mathbf{x}|\Theta) = \prod_{j=1}^{n} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.7}$$

Taking the logarithm of this probability gives

$$l(\Theta) = \log(p(\mathbf{x}|\Theta)) = \sum_{j=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right) \tag{4.8}$$

This is known as the log-likelihood function. To find the maximum likelihood solutions one must calculate the derivatives of $l(\Theta)$ with respect to $\pi_k, \boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ (Aggarwal & Reddy 2014). The derivative with respect to $\boldsymbol{\mu}_k$ is given by:

$$\frac{\partial l}{\partial \boldsymbol{\mu}_k} = \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \times \frac{\partial \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\partial \boldsymbol{\mu}_k} \tag{4.9}$$

$$= \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \times \frac{\pi_k \partial \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\partial \boldsymbol{\mu}_k} \tag{4.10}$$

$$= \sum_{j=1}^{n} \frac{-\frac{1}{2}(-2\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_k) \times \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \tag{4.11}$$

$$= \sum_{j=1}^{n} \frac{\pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_k) \tag{4.12}$$

$$= \sum_{j=1}^{n} \gamma(z_{jk}) \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_j - \boldsymbol{\mu}_k) \tag{4.13}$$

We found the derivative of $\mathcal{N}$ with respect to $\boldsymbol{\mu}_k$ using The Matrix Cookbook (Petersen & Pedersen 2008). In particular we used Equation (86) which is as follows:

$$\frac{\partial}{\partial \mathbf{s}} (\mathbf{x} - \mathbf{s})^T \mathbf{W} (\mathbf{x} - \mathbf{s}) = -2\mathbf{W} (\mathbf{x} - \mathbf{s}) \tag{4.14}$$

This implies that

$$\frac{\partial \mathcal{N}}{\partial \boldsymbol{\mu}_k} = \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(\frac{1}{(2\pi)^{p/2}} \frac{1}{|\boldsymbol{\Sigma}_k|^{1/2}} \exp(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k)) \right) \tag{4.15}$$

$$= \frac{\partial}{\partial \boldsymbol{\mu}_k} \left(-\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \right) \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.16}$$

$$= -\frac{1}{2} \times -2\boldsymbol{\Sigma}_k^{-1} (\mathbf{x} - \boldsymbol{\mu}_k) \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{4.17}$$

Setting (4.13) equal to 0 and multiplying by $\boldsymbol{\Sigma}_k$ we get

$$\boldsymbol{\mu}_k = \frac{\sum_{j=1}^{n} \gamma(z_{jk}) \mathbf{x}_j}{\sum_{j=1}^{n} \gamma(z_{jk})} \tag{4.18}$$

where $\gamma(z_{jk})$ is the same probability as that defined in Equation (4.6). Therefore we can think of the mean μ_k for the k-th Gaussian component as the weighted average of all of the observations in the dataset $\mathbf{x}$. The weighting factor is $\gamma(z_{jk})$ which is the posterior probability that component k generated observation $\mathbf{x}_j$. Similarly, we can calculate the

derivative of the log-likelihood function with respect to $\mathbf{\Sigma}_k$:

$$\frac{\partial l}{\partial \mathbf{\Sigma}_k} = \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i)} \times \frac{\partial \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i)}{\partial \mathbf{\Sigma}_k} \tag{4.19}$$

$$= \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i)} \times \frac{\pi_k \partial \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\partial \mathbf{\Sigma}_k} \tag{4.20}$$

$$= \sum_{j=1}^{n} \frac{-\frac{1}{2}\left(\mathbf{\Sigma}_k^{-1} - \mathbf{\Sigma}_k^{-1}(\boldsymbol{x}_j - \boldsymbol{\mu}_k)(\boldsymbol{x}_j - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}\right) \times \pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \mathbf{\Sigma}_i)} \tag{4.21}$$

$$= -\frac{1}{2} \sum_{j=1}^{n} \gamma(z_{jk}) \left(\mathbf{\Sigma}_k^{-1} - \mathbf{\Sigma}_k^{-1}(\boldsymbol{x}_j - \boldsymbol{\mu}_k)(\boldsymbol{x}_j - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}\right) \tag{4.22}$$

We found the derivative of $\mathcal{N}$ with respect to $\mathbf{\Sigma}_k$ using The Matrix Cookbook (Petersen & Pedersen 2008). In particular we used Equations (58) and (61) which are as follows:

$$\frac{\partial \det(\mathbf{x}^k)}{\partial \mathbf{x}} = k \det(\mathbf{x}^k)\mathbf{x}^{-T} \tag{4.23}$$

$$\frac{\partial a^T \mathbf{x}^{-1} b}{\partial \mathbf{x}} = -\mathbf{x}^{-T} a b^T \mathbf{x}^{-T} \tag{4.24}$$

These identities imply

$$\frac{\partial \mathcal{N}}{\partial \mathbf{\Sigma}_k} = \frac{\partial}{\partial \mathbf{\Sigma}_k}\left(\frac{1}{(2\pi)^{p/2}} \frac{1}{|\mathbf{\Sigma}_k|^{1/2}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k))\right) \tag{4.25}$$

$$= \frac{\partial}{\partial \mathbf{\Sigma}_k}\left(\frac{1}{|\mathbf{\Sigma}_k|^{1/2}}\right) \frac{1}{(2\pi)^{p/2}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)) \tag{4.26}$$

$$+ \frac{\partial}{\partial \mathbf{\Sigma}_k}\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)\right) \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{4.27}$$

$$= -\frac{1}{2} \frac{1}{|\mathbf{\Sigma}_k|^{1/2}} \mathbf{\Sigma}_k^{-1} \frac{1}{(2\pi)^{p/2}} \exp(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)) \tag{4.28}$$

$$+ \frac{1}{2} \mathbf{\Sigma}_k^{-1}(\mathbf{x} - \boldsymbol{\mu}_k)^T (\mathbf{x} - \boldsymbol{\mu}_k) \mathbf{\Sigma}_k^{-1} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{4.29}$$

$$= -\frac{1}{2}\left(\mathbf{\Sigma}_k^{-1} - \mathbf{\Sigma}_k^{-1}(\boldsymbol{x}_j - \boldsymbol{\mu}_k)(\boldsymbol{x}_j - \boldsymbol{\mu}_k)^T \mathbf{\Sigma}_k^{-1}\right) \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \mathbf{\Sigma}_k) \tag{4.30}$$

Setting (4.18) equal to zero and multiplying by $\mathbf{\Sigma}_k^2$, we get

$$\mathbf{\Sigma}_k = \frac{\sum_{j=1}^{n} \gamma(z_{jk})(\mathbf{x}_j - \boldsymbol{\mu}_k)(\mathbf{x}_j - \boldsymbol{\mu}_k)^T}{\sum_{j=1}^{n} \gamma(z_{jk})} \tag{4.31}$$

In a similar fashion to $\boldsymbol{\mu}_k$, we can think of $\mathbf{\Sigma}_k$ as the weighted average of the observations in the dataset with the weights being the conditional probability of each observation

being generated by a certain component. Finding the derivative of the log-likelihood function with respect to the mixing probabilities $\boldsymbol{\pi}_k$ is trickier because as we discussed earlier they must be positive and sum to 1. To get around this we can use a Lagrange multiplier. This means that we now want to maximise the following equation:

$$f = \log(p(\mathbf{x}|\Theta)) + \lambda \sum_{k=1}^{K} (\pi_k - 1) \tag{4.32}$$

We do this once again by differentiating

$$\frac{\partial f}{\partial \pi_k} = \frac{\partial}{\partial \pi_k} \left(\sum_{j=1}^{n} \log \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) + \lambda \sum_{i=1}^{K} (\pi_k - 1) \right) \tag{4.33}$$

$$= \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \times \frac{\partial \sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\partial \pi_k} + \lambda \tag{4.34}$$

$$= \sum_{j=1}^{n} \frac{1}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} \times \frac{\partial \pi_k}{\partial \pi_k} \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) + \lambda \tag{4.35}$$

$$= \sum_{j=1}^{n} \frac{\mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} + \lambda \tag{4.36}$$

$$= \sum_{j=1}^{n} \frac{\gamma(z_{jk})}{\pi_k} + \lambda \quad \Rightarrow \pi_k = \frac{-\sum_{j=1}^{n} \gamma(z_{jk})}{\lambda} \tag{4.37}$$

Summing over k and employing the constraint $\sum_{k=1}^{K} \pi_k = 1$ gives

$$1 = \frac{-n}{\lambda} \quad \Rightarrow \lambda = -n \tag{4.38}$$

Substituting for λ into (4.37)

$$\pi_k = \frac{\sum_{j=1}^{n} \gamma(z_{jk})}{n} \tag{4.39}$$

This can be seen as the average proportion of the overall dataset that the component accounts for. It is clear that $\gamma(z_{jk})$ rely on $\pi_k, \boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ whilst the maximum likelihood solutions to these parameters rely on $\gamma(z_{jk})$. This means that the task of maximising the log-likelihood function of the Gaussian mixture model is a very difficult one. Luckily there is an algorithm that finds a local optimum for this problem (Dempster et al. 1977).

4.2 Expectation Maximisation (EM) Algorithm

Before beginning the clustering, one must choose the initial values of the parameters of each Gaussian component $\{\pi_k^0, \boldsymbol{\mu}_k^0, \boldsymbol{\Sigma}_k^0\}_{k=1}^{K}$. A common way of doing this is as follows:

- Set each $\pi_k^0 = \frac{1}{K}$.
- Set each $\boldsymbol{\mu}_k^0$ as a randomly chosen observation from $(\mathbf{x}_1, ..., \mathbf{x}_n)^T$.
- Set each $\boldsymbol{\Sigma}_k^0$ as the covariance matrix of $(\mathbf{x}_1, ..., \mathbf{x}_n)^T$.

Alternatively, one could run the k-means algorithm discussed in Chapter 2 and set the initial values of the parameters of the Gaussian components as follows:

- Set each π_k^0 as the proportion of observations assigned to cluster k, $\frac{|C_k|}{n}$ where $|C_k|$ denotes the number of observations assigned to cluster k by the k-means algorithm.
- Set each $\boldsymbol{\mu}_k^0$ as the final mean vector $\boldsymbol{\mu}_k$ of a cluster obtained from the k-means algorithm.
- Set each $\boldsymbol{\Sigma}_k^0$ as the covariance matrix of a cluster obtained from the k-means algorithm.

The latter method usually results in faster convergence of the EM algorithm since the initial parameter values are likely to be closer to the optimal values found by the EM algorithm. Algorithm 6 describes the steps involved in the EM algorithm (Aggarwal & Reddy 2014).

Algorithm 6: EM ALGORITHM

Input: A dataset $\mathbf{x}$, a set of unobserved latent variables $Z = \{\boldsymbol{z}_j\}_{j=1}^n$, a set of unknown parameters $\Theta = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ and number of clusters K.

Output: The partitioning of $\mathbf{x}$ into K clusters which are represented by the parameters of K multivariate Gaussian distributions.

1 Choose the initial values of the parameters of the K Gaussian components, $\Theta^{(0)} = \{\pi_k^0, \boldsymbol{\mu}_k^0, \boldsymbol{\Sigma}_k^0\}_{k=1}^K$ (using one of the above methods) and calculate the initial log-likelihood function $\log(p(\mathbf{x}|\Theta^{(0)}))$ using Equation (4.8).

2 **E-step**: Calculate the posterior probability that component k generated observation $\mathbf{x}_j$, $\gamma(z_{jk})$ for each $k \in \{1, ..., K\}$ and $j \in \{1, ..., n\}$ using the current parameters $\Theta^{(t)}$ and Equation (4.6).

3 **M-step**: Update the parameters $\Theta^{t+1} = \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$ using the current $\gamma(z_{jk})$'s. Update the $\boldsymbol{\mu}_k$'s first then use these to update the $\boldsymbol{\Sigma}_k$'s followed by the π_k's. Use Equations (4.18), (4.31) and (4.39) respectively.

4 Calculate the log-likelihood function $\log(p(\mathbf{x}|\Theta^{(t+1)}))$ given the updated parameters $\Theta^{(t+1)}$ using Equation (4.8).

5 **if** $\log(p(\boldsymbol{x}|\Theta^{(t+1)})) - \log(p(\boldsymbol{x}|\Theta^{(t)})) \geq \epsilon$ **then**

6 ⌊ go back to step 2

7 **else**

8 ⌊ **return** $\Theta^{(t+1)}$

To partition the data into clusters one can take the final parameters and use them to calculate the posterior probability of each observation being generated by each component and assign $\mathbf{x}_j$ to the component i such that $\{\gamma(z_{jk})\}_{k=1}^{K}$ is maximised at $k = i$.

We shall now prove that each iteration of the EM algorithm is guaranteed to never decrease the log-likelihood function. We will be following the approach used in (Aggarwal & Reddy 2014). The EM algorithm actually works by maximising a lower-bound of the log-likelihood function. To understand why, we must introduce an inequality (Jensen 1906).

Definition 4.2.1 (Jensen's inequality)**.** For K non-negative numbers $\{\pi_1, ..., \pi_K\}$ such that $\sum_{k=1}^{K} \pi_k = 1$, K arbitrary numbers $\{x_1, ..., x_K\}$ and a convex function f:

$$f\left(\sum_{k=1}^{K} \pi_k x_k\right) \leq \sum_{k=1}^{K} \pi_k f\left(x_k\right) \tag{4.40}$$

The opposite is true when f is concave:

$$f\left(\sum_{k=1}^{K} \pi_k x_k\right) \geq \sum_{k=1}^{K} \pi_k f\left(x_k\right) \tag{4.41}$$

The logarithm is a concave function meaning the second inequality is true. Consider an arbitrary distribution $q(\mathbf{z}_j)$ over the latent variables $\mathbf{z}_j$ such that $0 \leq q(\mathbf{z}_j) \leq 1$ for $j \in \{1, ..., n\}$ and $\sum_{\mathbf{z}_j} q(\mathbf{z}_j) = 1$. It follows that for any such $q(\mathbf{z}_j)$ the following holds (Aggarwal & Reddy 2014):

$$l(\Theta) = \log(p(\mathbf{x}|\Theta)) = \sum_{j=1}^{n} \log\left(\sum_{\mathbf{z}_j} p(\mathbf{x}_j, \mathbf{z}_j|\Theta)\right) \tag{4.42}$$

$$= \sum_{j=1}^{n} \log\left(\sum_{\mathbf{z}_j} q(\mathbf{z}_j) \frac{p(\mathbf{x}_j, \mathbf{z}_j|\Theta)}{q(\mathbf{z}_j)}\right) \tag{4.43}$$

$$\geq \sum_{j=1}^{n} \sum_{\mathbf{z}_j} q(\mathbf{z}_j) \log\left(\frac{p(\mathbf{x}_j, \mathbf{z}_j|\Theta)}{q(\mathbf{z}_j)}\right) \equiv \mathcal{L}(q, \Theta) \tag{4.44}$$

Clearly $\mathcal{L}$ which is a function of both $q(\mathbf{z}_j)$ and Θ is a lower bound for the log-likelihood function $l(\Theta)$. But how tight is this lower bound? Suppose that $q(\mathbf{z}_j) = p(\mathbf{z}_j|\mathbf{x}_j, \Theta)$. Clearly this satisfies the conditions $0 \leq q(\mathbf{z}_j) \leq 1$ for $j \in \{1, ..., n\}$ and $\sum_{\mathbf{z}_j} q(\mathbf{z}_j) = 1$.

Then

$$\mathcal{L}(q,\Theta) = \sum_{j=1}^{n}\sum_{\mathbf{z}_j} p(\mathbf{z}_j|\mathbf{x}_j,\Theta)\log\left(\frac{p(\mathbf{x}_j,\mathbf{z}_j|\Theta)}{p(\mathbf{z}_j|\mathbf{x}_j,\Theta)}\right) \tag{4.45}$$

$$= \sum_{j=1}^{n}\sum_{\mathbf{z}_j} p(\mathbf{z}_j|\mathbf{x}_j,\Theta)\log\left(p(\mathbf{x}_j|\Theta)\right) \tag{4.46}$$

$$= \sum_{j=1}^{n}\log\left(p(\mathbf{x}_j|\Theta)\right) = l(\Theta) \tag{4.47}$$

This indicates that for this choice of $q(\mathbf{z}_j)$, $\mathcal{L}(q,\Theta) = l(\Theta)$, maximising $\mathcal{L}(q,\Theta)$ meaning that the lower bound is indeed tight. Now let $\Theta^{(t)}$ denote the current values for the Gaussian component parameters. The E-step of the Expectation-Maximisation algorithm maximises $\mathcal{L}(q,\Theta^{(t)})$ with respect to $q(\mathbf{z}_j)$. We can denote this as

$$q^{(t)}(\mathbf{z}_j) = \arg\max_{q}\mathcal{L}(q,\Theta^{(t)}) = p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)}) \tag{4.48}$$

As we showed above, $\mathcal{L}(q,\Theta^{(t)})$ is maximised when $q(\mathbf{z}_j) = p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})$. Therefore the E-step merely consists of finding the posterior probabilities $p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})$. In the M-step, $\mathcal{L}(q^{(t)},\Theta)$ is maximised with respect to $\Theta^{(t)}$ in order to update the parameters $\Theta^{(t+1)}$:

$$\Theta^{(t+1)} = \arg\max_{\Theta}\mathcal{L}(q^{(t)},\Theta) \tag{4.49}$$

Substituting for $q^{(t)}(\mathbf{z}_j) = p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})$ into $\mathcal{L}(q^{(t)},\Theta)$ in Equation (4.45), we can write $\mathcal{L}(q^{(t)},\Theta)$ as:

$$\mathcal{L}(q^{(t)},\Theta) = Q(\Theta.\Theta^{(t)}) - \sum_{j=1}^{n}\sum_{\mathbf{z}_j} p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})\log\Big(p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})\Big) \tag{4.50}$$

where $Q(\Theta.\Theta^{(t)}) = \sum_{j=1}^{n}\sum_{\mathbf{z}_j} p(\mathbf{z}_j|\mathbf{x}_j,\Theta^{(t)})\log\left(p(\mathbf{x}_j,\mathbf{z}_j|\Theta)\right)$ is the expectation of the complete data log-likelihood, $\mathbb{E}_{\mathbf{z}}[\log(p(\mathbf{x},\mathbf{z}|\Theta^{(t)})]$. Note that this is so-called because $\mathbf{x}$ is known as the incomplete dataset whilst $\{\mathbf{x},\mathbf{z}\}$ is known as the complete dataset (Salakhutdinov 2011). In an ideal world we would use the log-likelihood function for the complete dataset but since the only information we have on $\mathbf{z}$ comes from the posterior probability $p(\mathbf{z}|\mathbf{x},\Theta)$ we must use the expectation instead. The term on the right is known as the negative entropy of q which has no dependence on Θ. It follows that the task of maximising $\mathcal{L}(q^{(t)},\Theta)$ is equivalent to maximising $Q(\Theta.\Theta^{(t)})$ and we can rewrite the M-step as:

$$\Theta^{(t+1)} = \arg\max_{\Theta} Q(\Theta.\Theta^{(t)}) \tag{4.51}$$

It is clear that both the E and M-steps of the EM algorithm maximise $\mathcal{L}(q,\Theta)$ and that following the E-step we have (Aggarwal & Reddy 2014)

$$l(\Theta^{(t)}) = \mathcal{L}(q^{(t)},\Theta^{(t)}) \geq \mathcal{L}(q^{(t-1)},\Theta^{(t)}) \tag{4.52}$$

whilst after the M-step

$$\mathcal{L}(q^{(t)}, \Theta^{(t+1)}) \geq \mathcal{L}(q^{(t)}, \Theta^{(t)}) \tag{4.53}$$

This implies that

$$\mathcal{L}(q^{(t+1)}, \Theta^{(t+1)}) \geq \mathcal{L}(q^{(t)}, \Theta^{(t)}) \tag{4.54}$$

$$\Rightarrow \log(p(\mathbf{x}|\Theta^{(t+1)})) \geq \log(p(\mathbf{x}|\Theta^{(t)})) \tag{4.55}$$

So each iteration of the EM algorithm is guaranteed to monotonically increase this lower bound of the log-likelihood function by updating the parameters of the Gaussian components so that the actual log-likelihood function is monotonically increased as well. This also means that the algorithm is guaranteed to find a local maximum, the value of which is obviously dependent on the initial choice of parameter values $\Theta^{(0)}$.

4.2.1 Connection to the k-means algorithm

There is a link between the EM algorithm and the k-means algorithm. The k-means algorithm can be derived as a limit of the EM algorithm as follows (Bishop 2006). Assume that each component has the same covariance matrix given by $\epsilon\mathbf{I}$ where ϵ is a constant variance parameter and $\mathbf{I}$ is the $p \times p$ identity matrix. This implies that

$$\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \epsilon\mathbf{I}) = \frac{1}{(2\pi\epsilon)^{\frac{p}{2}}} \exp(-\frac{1}{2\epsilon}(\mathbf{x}_j - \boldsymbol{\mu}_k)^T(\mathbf{x}_j - \boldsymbol{\mu}_k)) \tag{4.56}$$

The EM algorithm can then be simplified as the covariance matrix of each component no longer needs to be estimated. The posterior probability $\gamma(z_{jk})$ of observation $\mathbf{x}_j$ being generated by component k can be written as

$$\gamma(z_{jk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(\mathbf{x}_j \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)} = \frac{\pi_k \exp(-\frac{1}{2\epsilon}(\mathbf{x}_j - \boldsymbol{\mu}_k)^T(\mathbf{x}_j - \boldsymbol{\mu}_k))}{\sum_{i=1}^{K} \pi_i \exp(-\frac{1}{2\epsilon}(\mathbf{x}_j - \boldsymbol{\mu}_i)^T(\mathbf{x}_j - \boldsymbol{\mu}_i))} \tag{4.57}$$

Consider the limit $\epsilon \to 0$. It follows that the posterior probabilities will go to zero for each k except $k = i$ for which $\gamma(z_{ji}) \to 1$. This occurs independently of the mixing probabilities π_k so long as none of them have value zero. So the E-step of the EM algorithm becomes a case of calculating the $\gamma(z_{jk})$'s, all bar one of which will be zero. The maximum likelihood solution for $\boldsymbol{\mu}_k$ simplifies to Equation (2.4) whilst that of the mixing probabilities π_k simplifies to the proportion of observations assigned to cluster k. Meanwhile the expectation of the complete data log-likelihood $Q(\Theta.\Theta^{(t)})$ becomes

$$Q(\Theta.\Theta^{(t)}) \to -\frac{1}{2}\sum_{k=1}^{K}\sum_{j=1}^{|C_k|}\sum_{l=1}^{p}(x_{jl} - \mu_{kl}^{\text{new}})^2 + c \tag{4.58}$$

where c is constant. Therefore the M-step of the EM algorithm which maximises $Q(\Theta.\Theta^{(t)})$ is equivalent to minimising the distortion for the k-means algorithm where distortion is defined in Equation (2.7). Once the algorithm converges one can assign each $\mathbf{x}_j$ to component i such that $\gamma(z_{ji}) = 1$. Therefore we achieve a hard clustering, i.e. each observation is assigned to exactly one cluster which in this case is that with the closest mean.

4.3 Advantages and disadvantages

The main advantage of the Gaussian mixture model combined with the EM algorithm is that it allows one to cluster datasets in which each cluster has unconstrained covariance structure (Schubert 2017). This widens massively the range of datasets that we can perform cluster analysis on. Although our implementation assigns each observation to the component which maximises the posterior probability, this does not always have to be the case. It can be useful to use soft clustering in which observations can belong to more than one cluster. Since the EM algorithm provides the probability of an observation being generated by each component, there could arise a scenario in which an observation is equally likely to come from two components. In this case, one could assign the observation to both clusters. Another advantage of the Gaussian mixture model is that the EM algorithm is relatively efficient. The EM algorithm has computational complexity $\mathcal{O}(K^2n^2p^2i)$ where i is the number of iterations. Whilst this is higher than the previous two clustering methods, a suitable choice of initial parameters can result in the algorithm converging quickly.

Some of the disadvantages of the Gaussian mixture model and EM algorithm are as follows: the EM algorithm only finds a local maximum of the log-likelihood function (Dempster et al. 1977). There could be many local maxima but the EM algorithm will is not guaranteed to find the global maximum. Similar to the k-means algorithm, it requires the number of clusters K to be pre-specified. This can be tricky to estimate if the dataset is large. Solutions to this problem include the elbow method which we discussed in Chapter 2. Another issue is that the log-likelihood can have singularities (Bishop 2006). Consider a scenario in which all of the components have covariance matrix given by $\Sigma_k = \sigma_k^2\mathbf{I}$ where $\mathbf{I}$ is the $p \times p$ identity matrix. Furthermore, suppose that one of the components has mean vector $\boldsymbol{\mu}_i = \mathbf{x}_j$ where $\mathbf{x}_j$ is an observation in the dataset. Then the log-likelihood function will contain a term given by

$$\mathcal{N}(\mathbf{x}_j|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \mathcal{N}(\mathbf{x}_j|\mathbf{x}_j, \sigma_i^2\mathbf{I}) = \frac{1}{(2\pi)^{\frac{p}{2}}}\frac{1}{\sigma_i^p} \tag{4.59}$$

Consider the limit $\sigma_i^p \to 0$. This causes both 4.59 and the log-likelihood function to go to infinity. This means that maximising the log-likelihood function is not always a well-defined problem. One solution is to detect when a component is collapsing onto an observation and reset the mean as a randomly chosen value whilst resetting the covariance matrix as a large value.

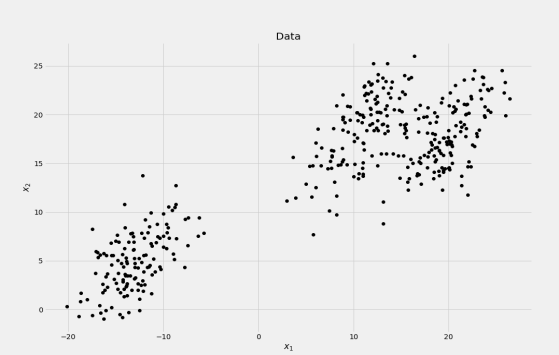
4.4 An example of Gaussian mixture model clustering

We shall now have a brief look at the use of the Gaussian mixture model on a toy dataset. Again we have made use of the Python function *sklearn.datasets.make_blobs* to generate a two-dimensional dataset with parameters $n = 420, K = 3,\ \boldsymbol{\mu}_1 = (-13, 5),\ \boldsymbol{\mu}_2 =$

$(10, 18)$, $\boldsymbol{\mu}_3 = (20, 18)$ and $\boldsymbol{\Sigma}_k = \begin{pmatrix} 1.5 & 0 \\ 0 & 1.5 \end{pmatrix}$ for all three blobs along with another argument *random_state* = 20 which determines the random number generation method. To give the data a more interesting covariance structure we have stretched the dataset by multiplying it by a randomly-generated 2×2 matrix. We have run the k-means algorithm on the dataset to find suitable values for the initial component means $\boldsymbol{\mu}_k^0$. We have set the initial covariance matrix values $\boldsymbol{\Sigma}_k^0$ to $\begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}$ for each $k = 1, 2, 3$ in order to illustrate the ability of EM algorithm to find a good approximate to the actual covariance of the dataset. We have set the initial mixing probabilities equal to $\frac{1}{3}$ since *sklearn.datasets.make_blobs* assigns an equal number of observations to each cluster where possible. We have used the sklearn class *sklearn.mixture.GaussianMixture* to implement the Gaussian mixture model. This automatically uses the EM algorithm to to find a local maximum of the log-likelihood function and hence a local optimum to the component parameters. We input the above initial component parameter values using the arguments *means_init*, *weights_init* and *precisions_init*. Note that the precision is the inverse of the covariance matrix.

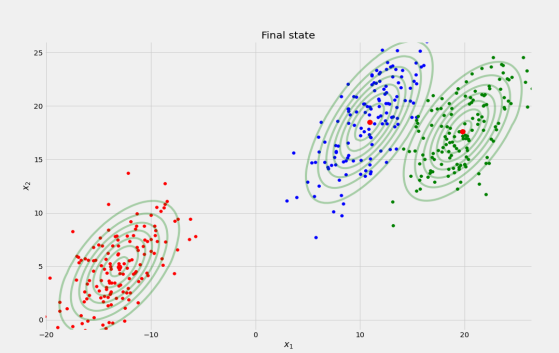
Figure 4.1 illustrates the dataset before it has been clustered along with a visualisation of the parameter initialisation, the final fit and a graph of the log-likelihood function against the number of iterations of the EM algorithm. The final results show that the final mean vectors from running the k-means algorithm which we used as initial values for the means of the Gaussian components were good estimates as they have not moved much during the running of the EM algorithm and appear to lie in the centres of the three clusters. The covariance matrices have changed as expected to capture the variance in the dataset. Looking at the graph of the log-likelihood function of the Gaussian mixture model, it appears to be as expected, increasing sharply over the first few iterations before flattening out after the fourth or fifth iteration, converging on a value of around 0.369. The different coloured dots on Figure 4.1c indicate the different cluster assignments. Looking at this, the clusters formed by algorithm 6 appear to be valid. In the next chapter we will be discussing some formal methods used to evaluate the results following a clustering.

Figure 4.1

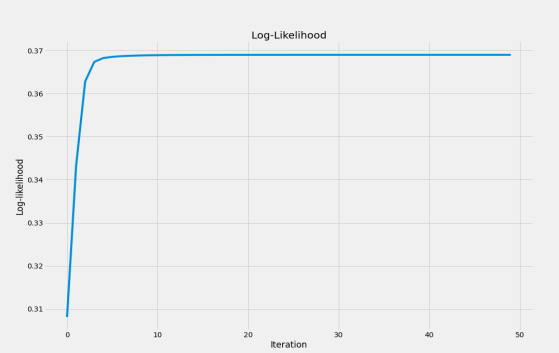


(a) Visualisation of a simple two-dimensional dataset.

(b) Visualisation of Gaussian components based on initial parameters Θ^0



(c) Visualisation of Gaussian components based on final parameters along with the final cluster assignments



(d) Graph of log-likelihood function against the number of iterations of the EM algorithm. Convergence occurs after roughly 8 iterations.

Chapter 5

Evaluating the results of cluster analysis

Once we have partitioned our dataset into clusters it is important to get an understanding of how "good" the clustering is. Indeed, evaluating the results of cluster analysis is an incredibly important part of the process and can be equally as tricky as the clustering itself (Pfitzner et al. 2009). So how do we decide what makes a clustering "good"? We can start by looking at the definition of cluster analysis which is the organisation of observations into different groups based on their features in such a way that observations within the same cluster have features more similar to each other than to those in different clusters. From this definition, it seems to safe to say that a "good" clustering should contain clusters which contain observations which are "close" to one another and the clusters themselves should be "far away" from one another. This summarises one of the main approaches to cluster evaluation which we call *internal evaluation*.

5.1 Internal Methods

In general, internal evaluation methods use only the information available in the dataset to give a single quality score. Some of them are the objective functions being optimised by a clustering model or algorithm (Feldman & Sanger 2006). This is almost the case for distortion which is equal to half of the within-cluster-variation, the objective function in the k-means algorithm. Of course, similar to the clustering methods themselves, the results of evaluation methods depend greatly on how they define a cluster. For example, density-based methods such as OPTICS define clusters as being an area of high density surrounded by an area of lower density (Schubert 2017). Therefore evaluation methods which define clusters similarly will usually give preference to clusterings produced by such methods over, say distance-based methods. We shall now discuss some examples of internal evaluation.

5.1.1 Silhouette coefficient

Definition 5.1.1 (Silhouette coefficient)**.** The silhouette coefficient utilises both the intra- and inter-cluster distances (Rousseeuw 1987). For a given observation $\mathbf{x}_j$ in cluster k let a_j be the mean distance between $\mathbf{x}_j$ and the other observations in its cluster. Similarly let b_j be the smallest mean distance between $\mathbf{x}_j$ and all other observations in a different cluster. The *average silhouette coefficient*, S is then given by:

$$S = \frac{\sum_{j=1}^{n} \frac{b_j - a_j}{\max(a_j, b_j)}}{n} \tag{5.1}$$

where

$$a_j = \frac{1}{|C_k| - 1} \sum_{l \in C_k, l \neq j} d(\mathbf{x}_j, \mathbf{x}_l) \quad \text{and} \quad b_j = \min_{i \neq k} \frac{1}{|C_i|} \sum_{l \in C_i,} d(\mathbf{x}_j, \mathbf{x}_l) \tag{5.2}$$

Any distance function can be used but the most commonly used functions are $d_{Euclidean}$ and $d_{Manhattan}$ which are defined in Chapter 1. $S \in [-1, 1]$ with a higher score indicating low intra-cluster distance and high inter-cluster distance hence indicating a good clustering. In general a value in the range (0.25,0.5] indicates a weak clustering , a value in the range (0.5,0.7] indicates a reasonable clustering whilst a value in the range (0.7,1] indicates a strong clustering (Schubert 2017). Values close to zero indicate that a lot of observations could be assigned to multiple clusters and negative values suggest that the clustering is very poor.

5.1.2 Davies–Bouldin index

Definition 5.1.2. The Davies–Bouldin index is similar to the silhouette coefficient in that it involves intra- and inter-cluster distances but in this case it is a ratio of the two (Davies & Bouldin 1979).

$$\text{DB} = \frac{1}{K} \sum_{k=1}^{K} \max_{i \neq k} \left(\frac{S_k + S_i}{M_{k,i}} \right) \tag{5.3}$$

where

$$S_k = \left(\frac{1}{|C_k|} \sum_{j \in C_k} |\mathbf{x}_j - \boldsymbol{\mu}_k|^2 \right)^{\frac{1}{2}} \quad \text{and } M_{k,i} = \left(\sum_{l=1}^{p} |\mu_{kl} - \mu_{il}|^2 \right)^{\frac{1}{2}} \tag{5.4}$$

where $|C_k|$ is the number of observations in cluster k and $\boldsymbol{\mu}_k$ is the mean vector for cluster k. DB $\in \mathcal{R}_{\geq 0}$ with a value closer to 0 indicating a better clustering. This makes sense since we want a clustering with a low scatter (S_k) and high separation ($M_{k,i}$).

5.2 External Methods

External methods compare the assigned cluster labels to the true cluster labels of the dataset (Schubert 2017). Of course, in the majority of real world datasets, the true cluster labels are not known and so these evaluation methods cannot always be used. However these methods can be used to compare any two clusterings of the same dataset and so they still have their uses. They can also be used to determine the strength of potential clustering methods. One issue that arises from clustering when we know the ground truth labels is that once we have our clusters, how do we know which true cluster each one corresponds to. To get around this, external methods tend to check whether each pair of observations which have the same true label have been clustered together by our clustering method (Pfitzner et al. 2009).

5.2.1 Purity

Definition 5.2.1. Purity is a measure of the level to which clusters contain observations with the same true label (Manning et al. 2008). Let C_k denote the set containing the indices of the observations assigned to cluster k and let T_k denote the set containing the indices of the observations with true label k. The *purity* of the clustering is then given by:

$$Purity = \frac{1}{n}\sum_{k=1}^{K} max_i|C_k \cap T_i| \tag{5.5}$$

Poor clusterings have purity close to 0 whilst a perfect clustering will have purity 1. This is because perfect clusterings will group observations together which have the same true label and so each C_k will be equal to a T_i. In reality, a clustering will rarely be perfect meaning not every C_k matches up to one C_i and so we match it with the T_i that shares the highest number of elements. This is the reason for the max in the equation.

5.2.2 Rand index

Definition 5.2.2. The *Rand index* is a measure of the similarity between the observations cluster assignments and their true labels (Rand 1971). It is computed as follows:

$$RI = \frac{TP + TN}{TP + FP + TN + FN} \tag{5.6}$$

where TP, FP, TN, FN are the respective numbers of true positives, false positives, true negatives and false negatives. For clusterings with $K > 2$ these refer to the number of correct pairwise assignments, i.e. TP is the number of pairs of observations that are assigned to the same cluster by the model and also share the same true label. Similarly FP is the number of pairs of observations that are assigned to the same cluster but have different true labels. FN is the number of pairs of observations that have the same true label but are not assigned to the same cluster. Lastly TN is the number of

pairs of observations that are not assigned to the same cluster and are also do not share the same true label. $RI \in [0, 1]$ with 1 indicating a perfect score. One can think of $TP + TN$ as the number of agreements between the clustering and the true labels whilst $FP + FN$ can be thought of as the number of disagreements between the two. Therefore RI can be interpreted as the proportion of pairs of observations that are agreed upon by the clustering and the true labels. Note that the the total number of pairs is $\binom{n}{2} = n(n-1)/2$ so the Rand index can alternatively be written as

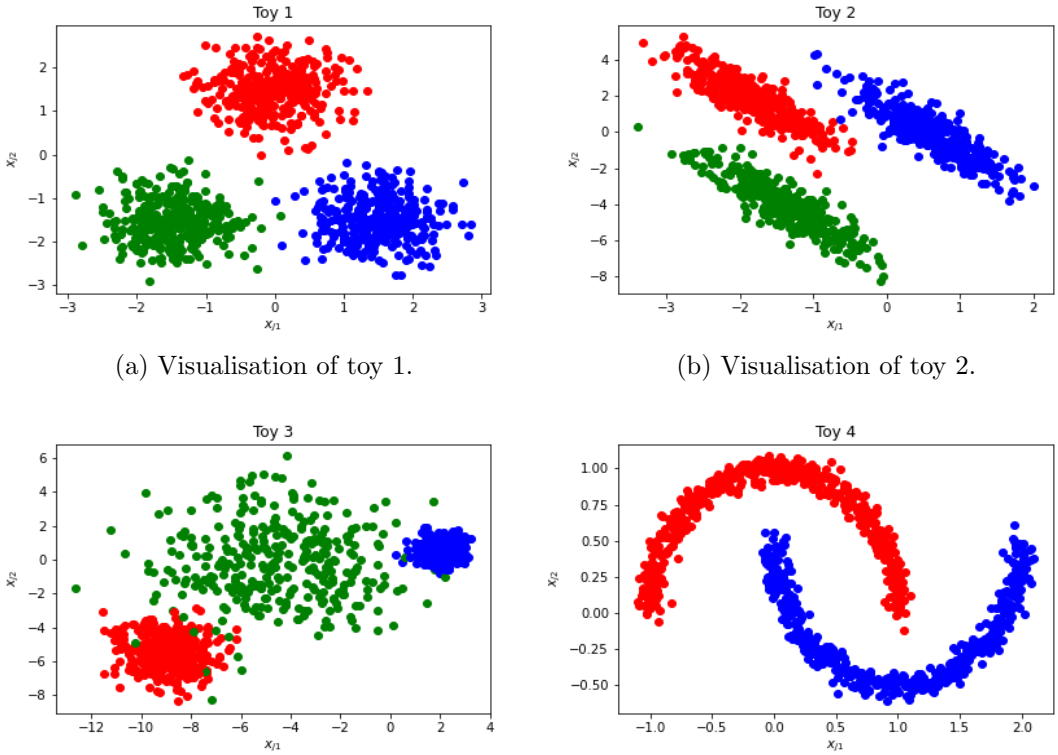
$$RI = \frac{TP + TN}{n(n-1)/2} \tag{5.7}$$

Chapter 6

Toy examples

We shall now take a look at how well our clustering methods perform on four toy datasets. We can also use the evaluation methods from the previous chapter to get an idea of how well each method does. So what do our toy datasets look like? For simplicity, we have generated the datasets to be 2-dimensional i.e. $k = 2$. This allows us to visualise the results easily. We have already seen toy 1 in Chapter 2 when discussing the elbow method. It has a simple structure with three distinct clusters. We have generated toy 2 using the same function *sklearn.datasets.make_blobs* with parameters $n = 1000, K = 3$ and $random\_state = 170$ before transforming it by multiplying it by the matrix $\begin{pmatrix} 0.4 & -0.4 \\ -0.4 & 1.4 \end{pmatrix}$. We have done this in order to give the clusters a more interesting covariance structure. Toy 3 has again be generated using *sklearn.datasets.make_blobs* with parameters $n = 1000, K = 3, \Sigma_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 2.5 & 0 \\ 0 & 2.5 \end{pmatrix}, \Sigma_3 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$ and $random\_state = 170$. The different covariance matrices result in the clusters being closer together with the cluster on the right almost completely overlapping with the central, large cluster. We have also given each cluster a different density to see how OPTICS performs. The final dataset, toy 4 is very different to the other three in that its clusters are non-convex in shape. We have used the function *sklearn.datasets.make_moons* with parameters $n = 1000$ and $noise = 0.05$ where the noise parameter determines the standard deviation of some Gaussian noise which has been added to the dataset. The two semi-circular clusters are interleaved meaning our clustering methods should struggle more with this dataset. The good thing about the sklearn functions is that they assign a true cluster label to each observation hence we will be able to use both internal and external methods to evaluate our clusterings. We have used the sklearn class *sklearn.cluster.KMeans* to implement the k-means algorithm. Similarly we have used the sklearn class *sklearn.cluster.OPTICS* to implement the OPTICS algorithm. We set the *cluster_method* parameter equal to "ξ" so that the ξ-method is used to extract the clusters. The default ξ value for this class is 0.05 so assume this is the value used unless otherwise stated. Lastly we have used the sklearn class *sklearn.mixture.GaussianMixture* for the Gaussian mixture model. This automatically uses the EM algorithm to to find

Figure 6.1



(a) Visualisation of toy 1.

(b) Visualisation of toy 2.

(c) Visualisation of toy 3.

(d) Visualisation of toy 4.

a local maximum of the log-likelihood function and hence a local optimum to the component parameters. We set the parameter *init_params* equal to "kmeans" so that the initial values of the parameters are based on the results of the k-means algorithm on the same dataset. Figure 6.1 illustrates each of our toy datasets, with the different colour dots indicating different cluster assignments. Figures 6.2 and 6.3 show the results of each clustering method. In the visualisations of the k-means and Gaussian mixture model clusterings the black dots represent the final mean vectors $\boldsymbol{\mu}_k$. Meanwhile the black dots in the visualisations of the clusterings produced by the OPTICS algorithm represent observations that have been labelled as noise/outliers. Table 6.1 details the results of the evaluation methods on each of our clusterings. Figure 6.4 contains the reachability plots drawn using the ordering produced by OPTICS on each of our toy dataset.

6.1 Toy 1 dataset

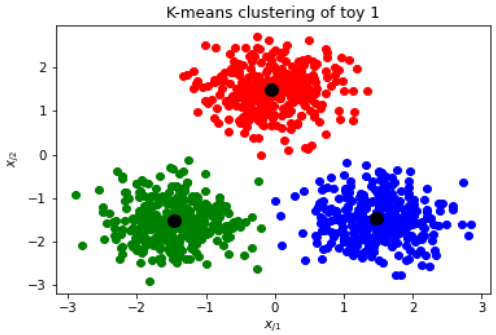
All of our clustering methods appear to perform well on toy 1. This is as expected since the clusters have a simple structure and are reasonably far apart from each other. Each cluster has an ovular shape meaning k-means performs well with Purity 0.999

meaning almost all clusters contain observations with the same true label. OPTICS also performs well since the clusters clearly match OPTICS' definition of a cluster as an area of high density surrounded by an area of lower density. The reachability plot in Figure 6.4a agrees with the results shown in Figure 6.2b. There are three identical dents in the reachability plot indicating three clusters of equal size and density. The Gaussian mixture model (using the EM algorithm) performs identically to the k-means algorithm. This makes sense since the two algorithms become very similar when the covariance matrix of each cluster is diagonal as mentioned in Chapter 4.

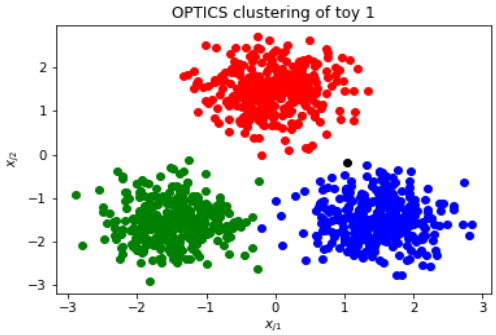
6.2 Toy 2 dataset

We designed the toy 2 dataset with the Gaussian mixture model in mind. In theory the GMM suits the dataset perfectly as each cluster looks as if it can be represented by a 2-dimensional Gaussian distribution. Each cluster has an obvious center and covariance structure and so the EM algorithm should be able to find good estimates of these parameters. The results confirm this, with almost all clusters containing observations with the same true label (Purity = 0.999). One thing that stands out looking at the results for toy 2 is that the clustering produced by the Gaussian mixture model has a smaller silhouette coefficient and higher Davies-Bouldin index than the clusterings produced by k-means and OPTICS. The reason for this is that we have used the Euclidean distance function ($d_{Euclidean}$) which means that this version of the silhouette coefficient gives preference to clusterings which contain spherical clusters. Since the Gaussian mixture model (and EM algorithm) has correctly clustered the dataset, the resulting clusters are not spherical and so according to this version of the silhouette coefficient, this is an inferior clustering. This highlights the importance of the choice of distance function in the silhouette coefficient. It also highlights the fact that we cannot judge a clustering model based solely off one internal evaluation method. If possible, it is much more useful to use an external method as well. Looking at Figure 6.2d, the k-means algorithm has clearly struggled with toy 2. It clearly struggles when the clusters are not spherical in shape which makes sense given its use of the Euclidean distance function. Figure 6.2e contains a number of black dots. These represent the observations which have been marked as outliers by the OPTICS algorithm. Whilst these have been incorrectly labelled, one can see why some of them have been labelled so. Several of these observations do appear to differ significantly from the other points. However, we should note that the majority of the dataset has been correctly clustered and so the clusters in toy 2 mostly agree with OPTICS' definition of a cluster. The reachability plot in Figure 6.4b agrees with the results shown in Figure 6.1b. There are three similar-looking dents in the reachability plot corresponding to the three clusters. All three dents have black points rising up on the right-hand side of the dent which represent the noise points. The blue dent has the highest number of these which agrees with Figure 6.2e.

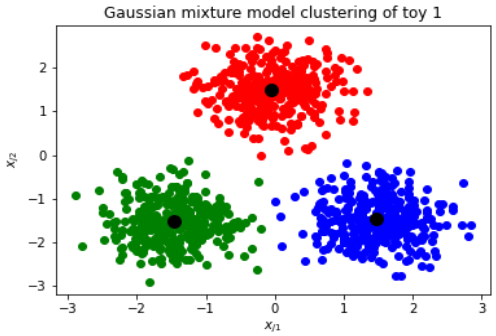
Figure 6.2



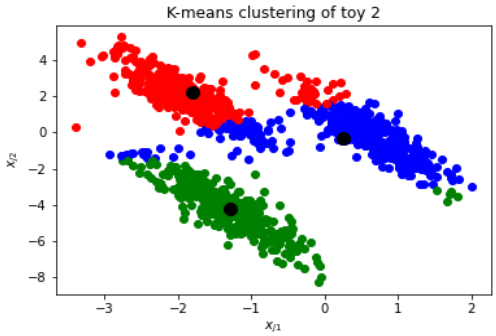
(a) Visualisation of results of k-means algorithm on toy 1.



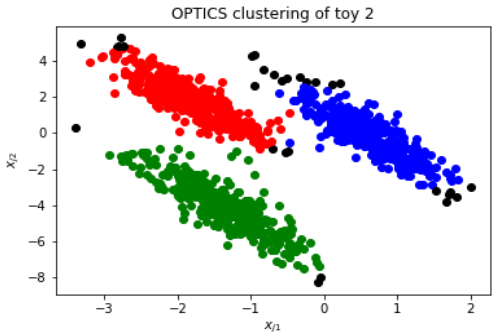
(b) Visualisation of results of OPTICS algorithm on toy 1. $\epsilon = \infty, MinPts = 28$.



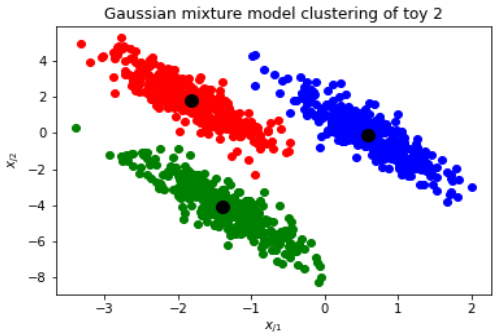
(c) Visualisation of results of Gaussian mixture model clustering on toy 1.



(d) Visualisation of results of k-means algorithm on toy 2.



(e) Visualisation of results of OPTICS algorithm on toy 2. $\epsilon = \infty, MinPts = 27$.



(f) Visualisation of results of Gaussian mixture model clustering on toy 2.

6.3 Toy 3 dataset

Despite the varying densities of the clusters in toy 3 OPTICS produces the clustering with the smallest purity. This is down to the fact that it has classed a lot of the observations

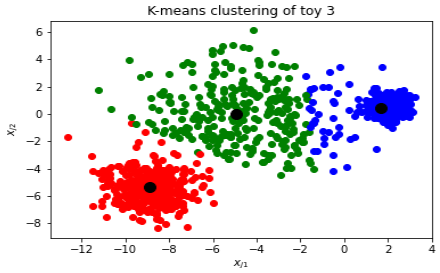
as outliers as shown in Figure 6.3b. Whilst it has performed poorly in this regard it does have the highest silhouette coefficient (0.689) and the lowest Davies-Bouldin index (0.481). This means that the clusters found by OPTICS are "good" in that they have low intra-cluster distance and high inter-cluster distance in general. The reachability plot for the OPTICS clustering in Figure 6.4c is rather jagged. This is due to the fact that we used a low value for $MinPts$ ($MinPts = 2$). The jaggedness occurs because fewer points have reachability-distance equal to their core-distance. Using a higher value led to the entire central cluster being labelled as noise due to its low density. We can also see that the deepest dent corresponds to the blue cluster meaning it has the highest density. This agrees with Figure 6.3b. Figure 6.3a suggests that the k-means performs slightly better than OPTICS although a number of observations in the top right have been clustered incorrectly. Clearly the k-means method has its limitations when the clusters overlap. This happens because the green cluster is much larger than the blue one and so some of the observations which should belong to the green cluster are actually closer to the centroid of the blue cluster hence they have been assigned to the wrong cluster. Looking at Figure 6.3c the Gaussian mixture model performs very well achieving a purity of 0.982. Clearly each cluster can be represented by a Gaussian component with unique mean and covariance. The structure of the different clusters is easily captured by the covariance matrices.

6.4 Toy 4 dataset

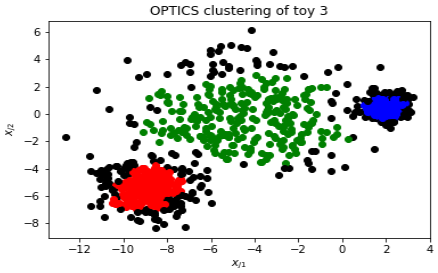
Toy 4 was designed to see how the clustering methods perform when the dataset contains clusters with non-convex shapes. Looking at Figures 6.3d, 6.3e and 6.3f, OPTICS clearly performs the best and this is backed up by the external evaluation methods. It achieves a purity of 1.000 meaning the cluster assignment for every observation matches its true label. No clustering method managed this on any of the other datasets. Again there is the issue of OPTICS producing the worst clustering according to the internal evaluation methods. Like the clustering produced by the Gaussian mixture model on toy 2, the clustering produced by OPTICS contains clusters which are not spherical in shape and so this clustering is penalised by both the silhouette coefficient and Davies-Bouldin index which use Euclidean distance to measure intra-cluster and inter-cluster distance. Figure 6.4d is interesting as the reachability plot is rather jagged. This has happened because every point has been clustered correctly with no outliers. In addition, none of the observations have very high reachability distance and so the range of reachability distances on the y-axis is much smaller for toy 4 than for the other datasets. This highlights the small differences in reachability distance among the clustered observations more than in the other plots. The k-means algorithm and Gaussian mixture model perform relatively poorly in comparison achieving Rand indexes of 0.625 and 0.756 respectively. This implies that both clusterings are rather dissimilar to the ground truth labels. Figures 6.3e and 6.3f indicate that they have incorrectly clustered the points at the ends of the clusters where the two semi-circular shapes interleave. This is due to the non-convex shape of the clusters which contradicts both methods' definition of a cluster. In the k-means

algorithm, each observation is assigned to the cluster with the closest mean vector using Euclidean distance and clearly looking at the final position of each centroid the observations where the two clusters interleave are closer to the centroids of the opposite cluster and have hence been assigned to the opposite cluster. With regards to the Gaussian mixture model, the EM algorithm finds a local maximum to the log-likelihood function, returning the optimal values of the parameters of the two Gaussian components. Each observation is then assigned to the component such that the posterior probability of the observation being generated by that component is maximised. Clearly given the two Gaussian components, the observations at the end of each cluster are more likely to be generated by the opposite component.

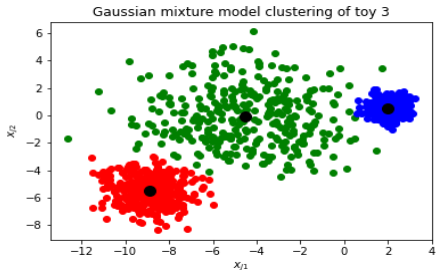
Figure 6.3



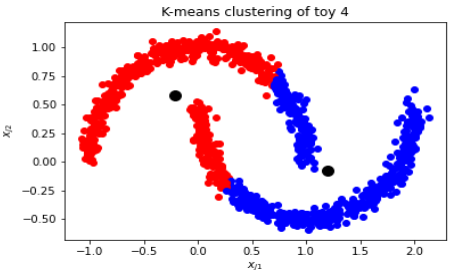
(a) Visualisation of results of k-means algorithm on toy 3.



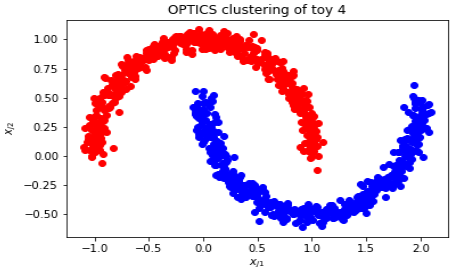
(b) Visualisation of results of OPTICS algorithm on toy 3. $\epsilon = \infty, MinPts = 2$.



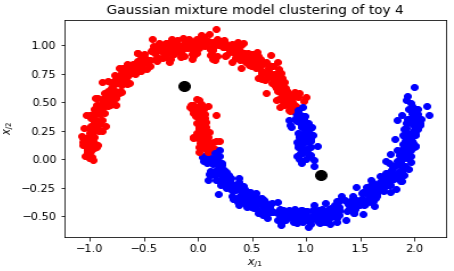
(c) Visualisation of results of Gaussian mixture model clustering on toy 3.



(d) Visualisation of results of k-means algorithm on toy 4.

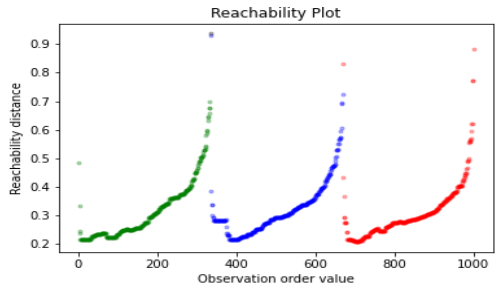


(e) Visualisation of results of OPTICS algorithm on toy 4. $\epsilon = \infty, MinPts = 19$.

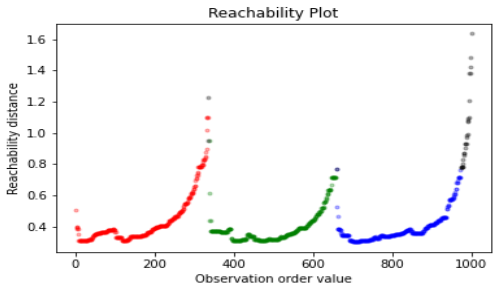


(f) Visualisation of results of Gaussian mixture model clustering on toy 4.

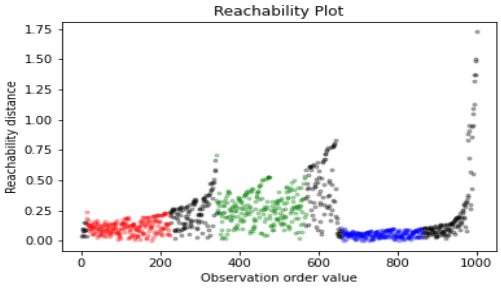
Figure 6.4



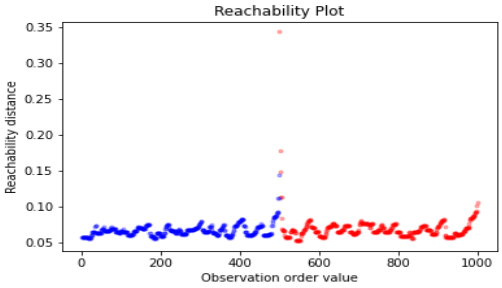
(a) Reachability plot produced using the OPTICS ordering of toy 1.



(b) Reachability plot produced using the OPTICS ordering of toy 2.



(c) Reachability plot produced using the OPTICS ordering of toy 3.



(d) Reachability plot produced using the OPTICS ordering of toy 4.

	Internal Methods		External methods	
Clustering	**Silhouette coefficient**	**Davies-Bouldin index**	**Purity**	**Rand-index**
Toy 1 k-means	0.705	0.399	0.999	0.999
Toy 1 OPTICS	0.705	0.399	0.998	0.997
Toy 1 GMM	0.705	0.399	0.999	0.999
Toy 2 k-means	0.524	0.657	0.882	0.862
Toy 2 OPTICS	0.529	0.640	0.981	0.971
Toy 2 GMM	0.497	0.707	0.999	0.999
Toy 3 k-means	0.644	0.574	0.941	0.927
Toy 3 OPTICS	0.689	0.481	0.852	0.837
Toy 3 GMM	0.622	0.584	0.982	0.976
Toy 4 k-means	0.491	0.779	0.750	0.625
Toy 4 OPTICS	0.335	1.156	1.000	1.000
Toy 4 GMM	0.466	0.830	0.858	0.756

Table 6.1: Table showing the results of different evaluation methods on our clusterings.

Chapter 7

Principal component analysis (PCA)

When dealing with high-dimensional data, it is often difficult to visualise the dataset as a whole. Therefore it is useful to find a lower-dimensional representation of the dataset ($p = 2$ or 3 allows us to plot easily) which still captures the majority of the variation within the dataset and so retains most of the information present in the original dataset. Not only can this assist with visualising the dataset, it can also help make clustering methods much more efficient. *Principal component analysis* (PCA) involves the orthogonal projection of a dataset onto a lower dimensional linear subspace which we will refer to as the principal subspace in such a way that the variance of the projected dataset is maximised (Bishop 2006). One method of PCA is formulated as follows (Hotelling 1933). Denote the desired dimensionality as m. For simplicity we shall first assume that $m = 1$. So we wish to orthogonally project our dataset $\mathbf{x}$ onto a one-dimensional linear space. Denote the direction of this space as $\boldsymbol{\gamma}_1$ and let it be a unit vector. Each observation $\mathbf{x}_j$ can be transformed onto this one-dimensional space as $\boldsymbol{\gamma}_1^T \mathbf{x}_j$. The *mean* and *variance* of the projected dataset are then given respectively by:

$$\boldsymbol{\gamma}_1^T \bar{\mathbf{x}} = \boldsymbol{\gamma}_1^T \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \quad \text{and} \quad \boldsymbol{\gamma}_1^T \mathbf{S} \boldsymbol{\gamma}_1 = \frac{1}{n} \sum_{j=1}^{n} (\boldsymbol{\gamma}_1^T \mathbf{x}_j - \boldsymbol{\gamma}_1^T \bar{\mathbf{x}})^2 \tag{7.1}$$

Here $\bar{\mathbf{x}}$ is the sample mean of the dataset and $\mathbf{S}$ is the covariance matrix of the dataset. Since we wish to maximise the variance of the projected dataset, we therefore need to maximise $\boldsymbol{\gamma}_1^T \mathbf{S} \boldsymbol{\gamma}_1$. We need to make this a constrained maximisation due to $\boldsymbol{\gamma}_1$ being a unit vector, i.e. $||\boldsymbol{\gamma}_1|| = \boldsymbol{\gamma}_1^T \boldsymbol{\gamma}_1 = 1$. To do this, we use a Lagrange multiplier, λ_1. Define

$$P(\boldsymbol{\gamma}_1) = \boldsymbol{\gamma}_1^T \mathbf{S} \boldsymbol{\gamma}_1 - \lambda_1 (\boldsymbol{\gamma}_1^T \boldsymbol{\gamma}_1 - 1) \tag{7.2}$$

Using Equations (69) and (85) from The Matrix Cookbook (Petersen & Pedersen 2008):

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a} \quad \text{and} \quad \frac{\partial}{\partial \mathbf{x}} \left(\mathbf{x}^T \mathbf{W} \mathbf{x} \right) = 2 \mathbf{W} \mathbf{x} \tag{7.3}$$

We get

$$\frac{\partial P}{\partial \boldsymbol{\gamma}_1} = 2\mathbf{S}\boldsymbol{\gamma}_1 - 2\lambda_1\boldsymbol{\gamma}_1 \tag{7.4}$$

Setting this equal to zero gives $\mathbf{S}\boldsymbol{\gamma}_1 = \lambda_1\boldsymbol{\gamma}_1$. So $\boldsymbol{\gamma}_1$ is an eigenvector of $\mathbf{S}$. Left-multiplying by $\boldsymbol{\gamma}_1^T$,

$$\mathrm{Var}[\boldsymbol{\gamma}_1^T\mathbf{x}_j] = \boldsymbol{\gamma}_1^T\mathbf{S}\boldsymbol{\gamma}_1 = \lambda_1\boldsymbol{\gamma}_1^T\boldsymbol{\gamma}_1 = \lambda_1 \tag{7.5}$$

Therefore the variance of the projected dataset will be largest when we set $\boldsymbol{\gamma}_1$ equal to the eigenvector of $\mathbf{S}$ which corresponds to the largest eigenvector λ_1 (Bishop 2006). This eigenvector is called the first principal component. To find the subsequent principal components, one can simply choose each to be the direction which maximises the variance of the projected dataset out of all those orthogonal to the principal components already defined. So for the orthogonal projection of $\mathbf{x}$ onto an m-dimensional linear space, the principal components are defined by the m eigenvectors $\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_m$ of $\mathbf{S}$ corresponding to the m largest eigenvalues $\lambda_1, ..., \lambda_m$.

But how do we find the values of the eigenvectors and their corresponding eigenvalues? Well, when dealing with high-dimensional datasets this can be very tricky but the following decomposition is useful (Einbeck 2020). We have for each $l \in \{1, ..., m\}$, $\mathbf{S}\boldsymbol{\gamma}_l = \lambda_l\boldsymbol{\gamma}_l$. Note that each side of this equation is an m-dimensional vector and so we can bind these together to give:

$$(\mathbf{S}\boldsymbol{\gamma}_1, ..., \mathbf{S}\boldsymbol{\gamma}_m) = (\lambda_1\boldsymbol{\gamma}_1, ..., \lambda_m\boldsymbol{\gamma}_m) \tag{7.6}$$

$$\boldsymbol{S}(\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_m) = (\boldsymbol{\gamma}_1, ..., \boldsymbol{\gamma}_m)\begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_m \end{pmatrix} \tag{7.7}$$

$$\boldsymbol{S}\boldsymbol{\Gamma} = \boldsymbol{\Gamma}\boldsymbol{\Lambda} \Rightarrow \boldsymbol{S} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^{-1} = \boldsymbol{\Gamma}\boldsymbol{\Lambda}\boldsymbol{\Gamma}^T \tag{7.8}$$

This decomposition allows one to find the values of the eigenvalues and eigenvectors, though this can be rather arduous when m is large and so the task is normally done by a computer program.

One thing we must consider when doing PCA is how many principal components we want to calculate. We know that $\mathrm{Var}[\boldsymbol{\gamma}_l^T\mathbf{x}_l] = \lambda_l$ so what happens when we sum over these eigenvalues? $\sum_{l=1}^m \lambda_l = \mathrm{Tr}(\boldsymbol{\Lambda}) = \boldsymbol{\Gamma}^T\mathbf{S}\boldsymbol{\Gamma} = \boldsymbol{\Gamma}\boldsymbol{\Gamma}^T\mathbf{S} = \mathrm{Tr}(\mathbf{S}) \equiv \mathrm{TV}(\mathbf{x})$ This is called the *total variance* and so for each l,

$$\frac{\lambda_l}{\sum_{l=1}^m \lambda_l} = \frac{\mathrm{Var}[\boldsymbol{\gamma}_l^T\mathbf{x}_l]}{\mathrm{TV}(\mathbf{x})} \tag{7.9}$$

is the proportion of total variance explained (PVE) by the l^{th} principal component. So if we wanted to reduce the dimensionality of a dataset we could simply calculate the eigenvalues and their corresponding PVE selecting those such that the majority (e.g. 95%) of the variance of the dataset is captured.

Chapter 8

Cluster analysis of the MNIST dataset

8.1 MNIST dataset

The MNIST (Modified National Institute of Standards and Technology) dataset is a collection of 70,000 images of handwritten digits from 0-9. The dataset is one of the most widely used in the field of cluster analysis due to its simplicity, ease of use and the fact that the dataset contains the true values of all 70,000 digits that have been drawn. The images are grayscale and 28 x 28 pixels which means the database requires little storage and is computationally cheap to perform cluster analysis on. Figure 8.1 displays an example of an image from the MNIST dataset.

8.2 Cluster analysis of image data

Before using any clustering method on the MNIST dataset the images have to be converted to the right dimension. Each image is stored as a 28×28 matrix, the values of this matrix being grayscale intensities. This matrix needs to be transformed into a 1-dimensional 1×784 matrix so that each image represents one observation and each

Figure 8.1: MNIST image with true value 3.

pixel represents one feature. Each observation will then have 784 features ($p = 784$). The 28×28 matrix is transformed into the 1×784 matrix by concatenating its rows together. Sometimes the scale of two feature columns can vary massively and so it is always good practise to standardise them so that they have mean 0 and standard deviation 1. However in the case of grayscale image data every feature column has the same range, the integer interval $\{0, ...,255\}$ and so there is no need to normalise the MNIST dataset.

8.3 Applying our methods to the MNIST dataset

To aid in the visualisation of the results of our clustering methods on the MNIST dataset we have performed PCA on it. We are using $m = 2$ so that the results can easily be plotted on a graph. However the proportion of total variance explained by the first two principal components combined is only 18% so we will not be running our clustering methods on such a low dimensional projected dataset. Instead we will be running them on the projection of the dataset onto the first 75 principal components which in total capture 90% of the total variance of the dataset.

Once the dataset has been converted to our required dimensions, we can run the k-means algorithm as normal supplying just the dataset $\mathbf{x}$ and the number of clusters K. We have taken a sample of size 1000 from the MNIST dataset so $n = 1000$ whilst there are ten different digits so we set $K = 10$. As mentioned above we have performed PCA on the dataset with $m = 75$ so this changes the number of features to $p = 75$. We have written our own Python function which implements the k-means algorithm.

For the implementation of the OPTICS algorithm we are using the sklearn class *sklearn.cluster.OPTICS*. We have set the *cluster_method* parameter equal to "ξ" so that the ξ-method is used to extract the clusters. The values of ϵ and $MinPts$ and ξ are ∞, 17 and 0.05 (5% which is the default) respectively. We have run the OPTICS algorithm on the same sample of size 1000 as we ran the k-means algorithm on. We have tried the algorithm with a range of values for $MinPts$ and ξ and will discuss the best ones.

8.4 Gaussian mixture models for the MNIST dataset

We are again using sklearn for our implementation of the Gaussian mixture model and EM algorithm. The class *sklearn.mixture.GaussianMixture* uses the EM algorithm to find a local maximum of the log-likelihood function and hence a local optimum to the component parameters. All we provide as inputs are the transformed dataset and the number of clusters, $K = 10$.

8.5 Evaluation of our clusterings

Looking at Figure 8.2a and imagining for the time-being that the dots are all black, it appears that the dataset will be tricky to cluster as (projected onto the first two principal components) the dataset appears to form one large cluster which does not look easily separable. This is confirmed by our results. Figure 8.2a shows clusters which are very close to each other. According to Table 8.1 the k-means clustering achieved a purity of 0.603, the best of the three methods. This is good considering a random cluster assignment of this dataset would be expected to achieve a purity of around 0.1. However, its Davies-Bouldin index is 2.523 meaning the ratio of intra-cluster to inter-cluster distance is higher than we would like. Similarly its silhouette coefficient of 0.094 is close to zero which indicates that a lot of observations are between two clusters. This is understandable given that there are ten clusters. The Gaussian mixture model (and EM algorithm) performs next best in terms of purity (0.573). Again it scores poorly with the internal methods with the same silhouette coefficient of 0.094 and a slightly better Davies-Bouldin index of 2.431. This is most likely due to the fact that the clusters are quite close to each other in terms of Euclidean distance. This means that the average inter-cluster distance is not so different to the average intra-cluster distance. This would certainly seem to be the indication given by Figure 8.2b although note that this only gives a visualisation in terms of the first two principal components and so does not paint the whole picture given by the dataset. In terms of the external evaluation methods, OPTICS performs the worst. However this is due to the fact that it labelled 595 of the observations as outliers. This was the best clustering we could produce having run the algorithm for many different values of $MinPts$ and ξ. In terms of the internal methods, the OPTICS clustering was very good. The silhouette coefficient of 0.453$\approx$0.5 suggests a reasonable clustering. The internal methods were carried out on the results excluding the outliers and so the results may be a bit skewed. The reachability plot shown in Figure 8.2c shows ten rather thin-looking dents in reachability with varying depths corresponding to the varying density of each cluster. The outliers-noise points have been excluded in order to make it easier to visualise the clusters.

One thing that we can do is convert the final mean vectors back into matrices so that we can visualise the mean of each vector as a grayscale image. Since we have performed PCA on the dataset we must first perform the inverse transformation on each mean vector. Each observation has been transformed into an m-dimensional vector as follows:

$$(\gamma_1, ..., \gamma_m)^T \mathbf{x}_j = \mathbf{x'}_j \tag{8.1}$$

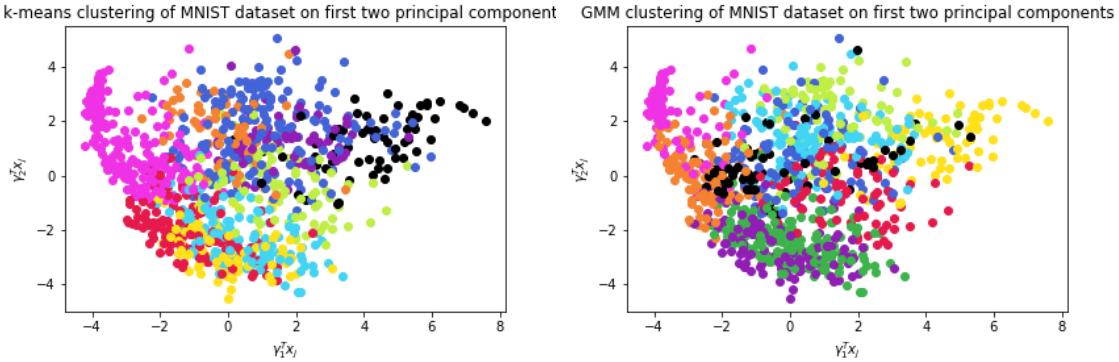
where we have $m = 75$ and $(\gamma_1, ..., \gamma_m)$ is the $n \times m$ matrix found by binding the eigenvectors λ_l of $\mathbf{S}$ together. Therefore each observation can be transformed back to (approximately) its original state as follows:

$$(\gamma_1, ..., \gamma_m)\mathbf{x'}_j \approx \mathbf{x}_j \tag{8.2}$$

Note that $\mathbf{x}_j$ is not recreated exactly because we lose information in the original projection. However, we still get a reasonable approximation to its original value. We can

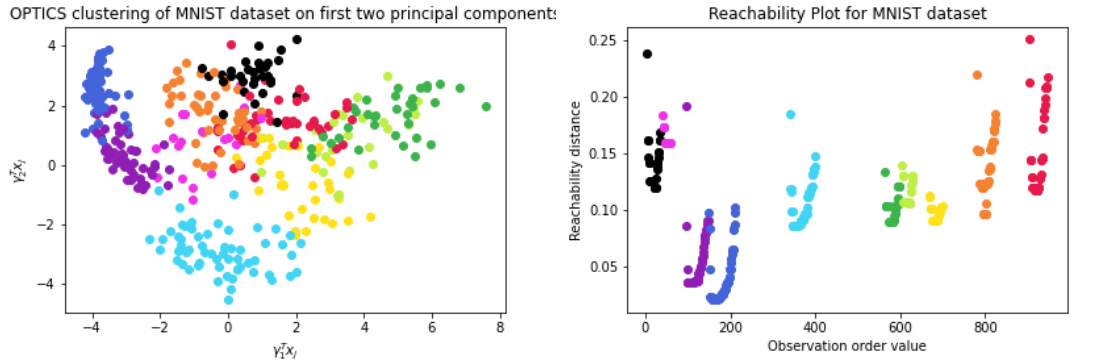
perform the same transformation on our final mean vectors $\boldsymbol{\mu_k}$ in order to view them as grayscale images. Figures 8.3 and 8.4 display the visualisations of the final mean vectors found by the k-means EM algorithms.

Figure 8.2: Visualisations of our clusterings on the first two principal components.



(a) Visualisation of k-means clustering of MNIST dataset on first two principal components.

(b) Visualisation of Gaussian mixture model clustering of MNIST dataset on first two principal components.



(c) Visualisation of OPTICS clustering of MNIST dataset on first two principal components.

(d) Reachability plot of the ordering produced by OPTICS on the MNIST dataset.

	Internal Methods		External methods	
Clustering	**Silhouette coefficient**	**Davies-Bouldin index**	**Purity**	**Rand-index**
k-means	0.094	2.523	0.603	0.869
OPTICS	0.453	0.856	0.493	0.649
GMM	0.094	2.431	0.573	0.880

Table 8.1: Table showing the results of different evaluation methods on our clusterings.

Looking at Figure 8.3, we can clearly see the most common true label in certain clusters.

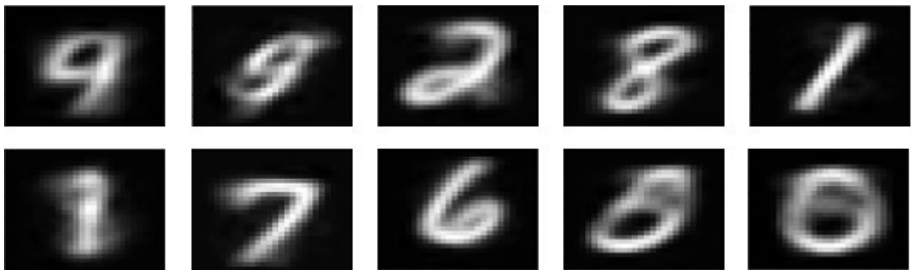


Figure 8.3: Visualisations of the final centroids found by the k-means algorithm on the MNIST dataset.

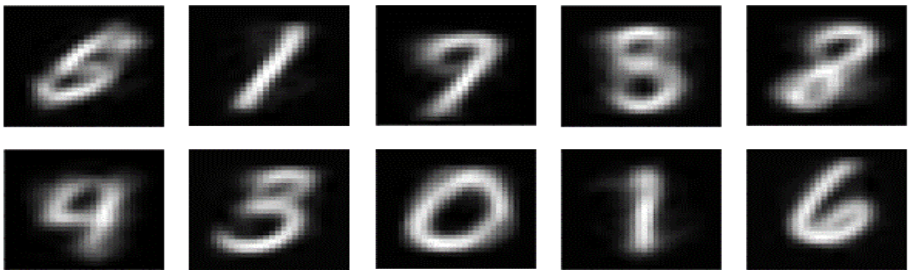


Figure 8.4: Visualisations of the final mean vectors found by the EM algorithm on the MNIST dataset.

The third, fourth and fifth images on the top row along with the second and third image on the bottom row clearly correspond to clusters containing mainly twos, eights, ones, sevens and sixes respectively. However, for the remaining images, it is much harder to tell which true label the majority of the observations in each cluster has. For example, the first image on the top row indicates that the majority of observations in that cluster had a true label of either four or nine. It is easy to see why these observations have been clustered together. Both numbers can look very similar when written down and so it is understandable that the k-means algorithm struggled to separate them into two different clusters. The last two images on the bottom row look as if they both contain a number of observations with true label zero. But these clusters could easily also contain fives, sixes and eights.

Similarly, the final mean vectors produced by the Gaussian mixture model and EM algorithm seem to represent some clusters containing observations with a mixture true labels. The second and fifth images on the top row and the second, third, fourth and fifth images on the bottom row seem to represent clusters containing mainly ones, eights, threes, zeros, ones and sixes respectively. However, the other images do not paint as clear a picture. For example, the third image on the top row looks as if the corresponding cluster contains mainly sevens and nines. Again this is understandable since when written down, the digits do not differ massively.

Chapter 9

Conclusion

In this report we have taken an in-depth look at three of the most commonly used methods in the field of cluster analysis. Each one differs from the next based on their definition of exactly what a cluster is with k-means' being a group of observations with low intra-cluster distance and high inter-cluster distance. Obviously, the results depend massively on the choice of distance function. On the other hand, the OPTICS algorithm defines a cluster as an area of high density surrounded by an area of low density. This density is measured by considering the number of observations within a particular ϵ-neighbourhood. Lastly, the Gaussian mixture model defines clusters as groups of observations which are most likely to come from the same multivariate Gaussian distribution. The clusters can then be represented by the parameters of said distribution. Whilst the steps taken in each method differ, the results produced can be very similar and there is even a special case of the Gaussian mixture model where the EM algorithm takes exactly the same steps as the k-means algorithm. There is no clustering method which we can definitively say is the best. There are many factors that need to be considered when deciding which method to use. These include the type of data being clustered, the size of the dataset, the type of distance function to use and how many clusters you want to separate the dataset into. Each clustering method has its advantages and disadvantages and these need to be carefully considered before choosing one. For example, if you are unfamiliar with the dataset then OPTICS may be the best method to use since it does not require you to pre-specify the number of clusters.

We have also looked at some of the methods used to evaluate the results of cluster analysis. In particular we have looked at two categories: internal methods and external methods. Internal methods use only the information available in the dataset to give a single quality score. For example, the silhouette coefficient looks at the mean distance between an observation and the other observations in its cluster and then finds the smallest mean distance between said observation and the observations in another cluster. It then subtracts the former from the latter and divides by the larger of the two before finding the average of this value across all n observations. These methods allow one to find out whether a clustering method produces clusters that have certain

properties that an ideal cluster should have, such as having low-intra cluster distance and high inter-cluster distance. The properties being checked obviously vary depending on the evaluation method used. External methods compare the assigned cluster labels to the ground truth labels of the dataset. For example, purity finds the number of observations in a cluster with the most common true label in that cluster before averaging this value across all clusters. Of course this is only useful when the ground truth labels are available and even when they are, this is only one clustering of the dataset. Who's to say that there isn't another clustering of the dataset which is equally as valid? It also begs the question of why we are clustering in the first place if we already have the ground truth labels. One answer to this is that it allows us to test out new clustering methods, to see how they perform.

We next had a look at how the three clustering methods performed on four, specially-generated 2-dimensional datasets. Toy 1 was a very simple dataset with three distinct, linearly separable clusters on which all three methods performed almost perfectly. Toy 2 was designed with the Gaussian mixture model in mind as the dataset has a more complex covariance structure. As expected, the Gaussian mixture model performed the best with the components' covariance matrices easily able to capture the variance of each cluster. Toy 3 was designed as more of a challenge to the clustering methods due to the fact that the clusters were closer together and one small but dense cluster was contained within another larger, less dense cluster. Despite this, all three methods performed well with OPTICS scoring the worst with the external evaluation methods only because it labelled so many observations as noise. Lastly, toy 4 was designed with OPTICS in mind. The non-convex shape of the clusters made it difficult for k-means and the Gaussian mixture model to cluster it accurately but as expected, OPTICS clustered it perfectly. The clusters were dense enough that the algorithm easily found them.

Before looking at how our methods performed on a real-world dataset we briefly discussed a dimensionality reduction technique called principal component analysis. This involves the orthogonal projection of a dataset onto a lower-dimensional linear space in such a way that the variance of the projected dataset is maximised. To do this, the eigenvectors and eigenvalues of the data covariance matrix are computed. This can be handy for making clustering methods more efficient whilst also making it easier to visualise their results.

The previous chapter saw us applying our clustering methods to a real world dataset known as the MNIST dataset. It contains 70,000 images of handwritten digits from 0 to 9. Due to the data being in image form, the 28×28 matrices representing the grayscale images first had to be converted to 784×1 vectors by concatenating the rows. On the whole, the clustering results were not as accurate as we would have liked. This is most likely down to the high dimensionality of the dataset and also the fact that there is a lot of variation within clusters. No two images of a number 4 will be the same and they can be quite different in-fact, depending on how one draws their fours. There are many

clustering methods out there which can cluster the MNIST dataset with almost perfect accuracy. For example Deep Adaptive Clustering (DAC) combines clustering with feature learning and uses a deep convolutional network achieving a purity of 97.75% on the MNIST dataset (Chang et al. 2017).

In our discussion so far we have assumed that all data being clustered is quantitative but what if the data is qualitative? Let's say that we are attempting to cluster a dataset with information on a group of people and one of the features is sex. How do we quantify the distance between "male" and "female"? Qualitative data contains categorical variables which can be split into two classes, ordinal and nominal (Raschka 2015). Ordinal features are categorical variables which can be ordered whilst nominal features cannot be ordered. For example consider a dataset containing information on a certain company's range of suitcases. An ordinal feature could be the "size" of the suitcases, S, M, L. The order of these values is obvious, S<M<L. A nominal feature could be the "colour" of the suitcases, red, black, silver. There is no clear order to these values. In the case of ordinal features the best method of circumvention is to map these values to integers which will maintain its ordinality whilst making it easier to define the distance between observations. In the case of nominal data one solution is to use one-hot encoding. This involves creating a dummy variable for each value within the feature (Raschka 2015). So in our suitcase example, the feature "colour" would be replaced by three features; "red", "black" and "silver". These new features are binary and so a black suitcase can be represented by "red"=0, "black"=1, "silver"=0. This means that a wide range of different types of data can be clustered, not just quantitative.

The methods we have described in this report were all invented in the 20^{th} century but what work is currently being done using cluster analysis? Well its currently being used in a vast array of fields. It is used in sequence analysis which involves the organisation of DNA into gene families (Remm et al. 2001). Hierarchical methods are particularly useful for this purpose since they provide clusterings with all possible numbers of clusters. Therefore one can not only find the gene families but also see how the families themselves are linked. Another current application of cluster analysis is in medical imaging. It can be used to distinguish between different types of tissue in PET scans (Filipovych et al. 2011). For example it can help decide whether a tumour is malignant or benign. Clustering models can be run extremely quickly in comparison to the time taken for a doctor to be able to look at an image and so people can get a diagnosis much quicker resulting in many lives being saved.

We should now have a reasonable understanding of the concepts behind and methods involved in the k-means algorithm, OPTICS algorithm and the Gaussian mixture model (and EM algorithm). We should also have an idea of how these methods are applied to datasets and how to visualise and evaluate the results. We have discussed three clustering methods but there are many more out there that are used in the real world for everything from market research to robotics. Cluster analysis is part of the larger field

of machine learning which is one of the most rapidly evolving industries in the world. It will only continue to grow as the computational power at our fingertips allows us to run machine learning models in less and less time.

Bibliography

Abu-Jamous, B., Fa, R. & Nandi, A. K. (2015), *Integrative Cluster Analysis in Bioinformatics*, Wiley.

Aggarwal, C. C. & Reddy, C. K. (2014), *Data clustering : algorithms and applications*, Chapman Hall.

Ankerst, M., Breunig, M. M., peter Kriegel, H. & Sander, J. (1999), Optics: Ordering points to identify the clustering structure, ACM Press, pp. 49–60.

Berkhin, P. (2002), 'Survey of clustering data mining techniques', *A Survey of Clustering Data Mining Techniques. Grouping Multidimensional Data: Recent Advances in Clustering.* **10**.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer.

Cattell, B. R. (1943), 'The description of personality: Basic traits resolved into clusters', *Journal of Abnormal and Social Psychology* **38**, 476–506.

Chang, J., Wang, L., Meng, G., Xiang, S. & Pan, C. (2017), Deep adaptive image clustering, *in* 'Proceedings of the IEEE International Conference on Computer Vision (ICCV)'.

Chauhan, R., Batra, P. & Chaudhary, S. (2014), 'A survey of density based clustering algorithms', *International Journal of Computer Science And Technology* **5**.

Davies, D. & Bouldin, D. (1979), 'A cluster separation measure', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **PAMI-1**, 224 – 227.

Dempster, A., Laird, N. & Rubin, D. (1977), 'Maximum likelihood from incomplete data via the em algorithm', *Journal of the Royal Statistical Society. Series B* **39**, 1–38.

Driver, H. & Kroeber, L. (1932), 'Quantitative expression of cultural relationships', *University of California Publications in American Archaeology and Ethnology* pp. 211–256.

Einbeck, J. (2020), 'Statistical methods 3 2020-2021 michaelmas and epiphany lecture notes'.

Ester, M., Kriegel, H.-P., Sander, J. & Xu, X. (1996), A density-based algorithm for discovering clusters in large spatial databases with noise, AAAI Press, pp. 226–231.

Feldman, R. & Sanger, J. (2006), *Core Text Mining Operations*, Cambridge University Press, p. 19–56.

Filipovych, R., Resnick, S. M. & Davatzikos, C. (2011), 'Semi-supervised cluster analysis of imaging data', *NeuroImage* **54**(3), 2185—2197.

Hotelling, H. (1933), 'Analysis of a complex of statistical variables into principal components', *Journal of Educational Psychology* **24**, 417–441.

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013), *An Introduction to Statistical Learning with Applications in R*, Springer Publishing.

Jensen, J. L. W. V. (1906), 'Sur les fonctions convexes et les inégualités entre les valeurs Moyennes'.

Kazi, A. & Kurian, D. T. (2014), 'A survey of data clustering techniques', *International Journal of Engineering Research Technology* **3**.

Ketchen, D. J. & Shook, C. L. (1996), 'The application of cluster analysis in strategic management research: An analysis and critique', **17**, 441–458.

Lengyel, T. (1994), 'On the divisibility by 2 of the stirling numbers of the second kind', *Fibonacci Quart* **32**(3), 194–201.

Lloyd, S. P. (1982), 'Least squares quantization in pcm', *IEEE Transactions on Information Theory* **28**, 129–137.

Macqueen, J. (1967), Some methods for classification and analysis of multivariate observations, *in* 'Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability', Vol. 1, pp. 281–297.

Manning, C. D., Raghavan, P. & Schütze, H. (2008), *Introduction to Information Retrieval*, Cambridge University Press.

Murphy, K. P. (2012), *Machine Learning: A Probabilistic Perspective*, MIT Press.

Petersen, K. B. & Pedersen, M. S. (2008), 'The matrix cookbook'. Version 20081110.

Pfitzner, D., Leibbrandt, R. & Powers, D. (2009), 'Characterization and evaluation of similarity measures for pairs of clusterings', *Knowl. Inf. Syst.* **19**, 361–394.

Rand, W. (1971), 'Objective criteria for the evaluation of clustering methods', *Journal of the American Statistical association* **66**(336), 846–850.

Raschka, S. (2015), *Python Machine Learning*, Packt Publishing.

Remm, M., Storm, C. E. V., Sonnhammer, E. L. L. & Karolinska, B. (2001), 'Automatic clustering of orthologs and in-paralogs from pairwise species comparisons', *J. Mol. Biol* pp. 1041–1052.

Rousseeuw, P. J. (1987), 'Silhouettes: a graphical aid to the interpretation and validation of cluster analysis', *Journal of computational and applied mathematics 20* pp. 53–65.

Saha, R., Tariq, M. T., Hadi, M. & Xiao, Y. (2019), 'Pattern recognition using clustering analysis to support transportation system management, operations, and modeling', *Journal of Advanced Transportation* .

Salakhutdinov, R. (2011), *Statistical Machine Learning, Lecture 6.*
URL: *http://www.utstat.toronto.edu/ rsalakhu/sta4273/notes/Lecture6.pdf*

Santini, M. (2016), *Advantages Disadvantages of k-Means and Hierarchical clustering (Unsupervised Learning)*, Uppsala University.
URL: *http://santini.se/teaching/ml/2016/Lect*$_1$0/10$c_U nsupervisedMethods.pdf$

Schubert, E. (2017), *Knowledge Discovery in Databases*, Heidelberg University.
URL: *https://dbs.ifi.uni-heidelberg.de/files/Team/eschubert/lectures/KDDClusterAnalysis17-screen.pdf*

Sneath, P. H. A. (1957), 'The application of computers to taxonomy', *Journal of General Microbiology* **17**, 201–226.

Sonagara, D. & Badheka, S. (2014), 'Comparison of basic clustering algorithms', *International Journal of Computer Science and Mobile Computing* **3**, 58–61.

Tryon, R. C. (1939), *Cluster Analysis: Correlation Profile and Orthometric (factor) Analysis for the Isolation of Unities in Mind and Personality*, Edwards Brothers.

Zubin, J. (1938), 'A technique for measuring like-mindedness', *The Journal of Abnormal and Social Psychology* **33**, 508–516.