Clustering Methods

Alex Dennington — 29534267

5th of January 2021

Contents

1 Abstract

This report will contain a study of the theory behind some fundamental clustering methods, their usage in modern statistics, and seeks to demonstrate their implementation in R using different datasets that will help to evaluate them. Additionally, more modern methods built from these fundamental methods will be briefly discussed.

2 Introduction

Clustering methods like K-Means clustering and Hierarchical clustering are unsupervised learning methods that are performed on datasets in order to extract meaningful information that can be used in a variety of ways. These ways include for linear mixed modelling and in business practices for pattern recognition and image processing. These fundamental clustering methods have a number of different ways of being implemented, with algorithms such as Lloyd's [2], MacQueen's [1] and Hartigan & Wong's [19] being shown for K-Means, and the agglomerative [3] and divisive [8] algorithms being shown for hierarchical clustering. A method derived from K-Means called Fuzzy Clustering [10, 11] is also shown, since this method allows for observations to belong to more than one cluster, and many real world applications appreciate this degree of flexibility; for example in gene analysis.[29] Another important aspect of clustering is deciding how many natural clusters the data has. This is particularly relevant for K-Means clustering since the algorithm requires that the number of clusters be prespecified, however, these methods can be applied to hierarchical clustering as well in order to ascertain how many clusters to select. In the appendices, the coding implementations of these methods are shown in R, with the use of additional packages. This section demonstrates how to perform these different clustering techniques, how to use different algorithms and initialisation procedures, and how to appraise the performance of the clustering model using indices like the Average Silhouette Coefficient[8] and the Dunn Index[10].

3 Unsupervised Learning

Clustering is a form of unsupervised learning, which is a set of methods for analysing datasets which only contain features of the observations and do not contain a response variable. Unsupervised learning methods can be used to simplify data into fewer variables to help with visualisation (like with Principal Component Analysis) or to look for patterns and subgroups among the features or observations (Clustering).

3.1 Clustering

There are many types of clustering, the most popular being Partitioning Methods, Hierarchical Clustering, Fuzzy Clustering and Density Based Clustering, but this report will only focus on the first three. Partitioning methods are ways of splitting the data up into distinct groups, the number being specified before the method is performed. A common type of partitioning method is K-means Clustering, where the obervations or features are split up into K distinct groups using a similarity measure. Hierarchical clustering works differently than K-means where instead of splitting up the data into groups, it creates a diagram called a dendrogram that looks very similar to a tree diagram. At the bottom of the diagram, observations (leaves) that connect to other observations via branches are considered very similar, where-as the higher up the diagram you go, the less similar they are, i.e. the earlier (lower down) the leaves and/or branches fuse, the more similar the observations or groups of observations are.

4 K-Means Clustering

This method seeks to divide the data into K-distinct groups which are as similar as possible. With this method, the number of groups have to be prespecified prior to performing the algorithm. Once finished, each observation will belong to one cluster. Since each of the observations are members of one cluster, the clusters are distinct and non-overlapping. The fact that data points belong to one cluster means this algorithm is refered to as a hard clustering algorithm. Figure 1 shows an example of a dataset that has been clustered with different values of K. In both cases, the datapoints have been put into K distinct groups. The aim of the method is to end up with K-distinct clusters where, of all the possible clusters, this assignment has the minimum total within-cluster variation.[1] In other words, to solve the equation:

$$\underset{C_1,\ldots,C_k}{\text{minimize}} \quad \left\{ \sum_{k=1}^{K} W(C_k) \right\}$$

where $W(C_k)$ is the within-cluster variation for cluster k. This is an NP-Hard problem. There are a few ways to define within-cluster variation. The most common choice is *squared euclidian distance.* This is

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

where x_{ij} is the jth feature of the ith point in cluster k. $|C_k|$ gives the total number of data points in cluster k.

4.1 Algorithms

Algorithms exist that can give a locally optimal solution to this problem, however they are not guarenteed to provide the globally optimal solution. Therefore, they are known as greedy algorithms. The most widely used algorithms are as follows:

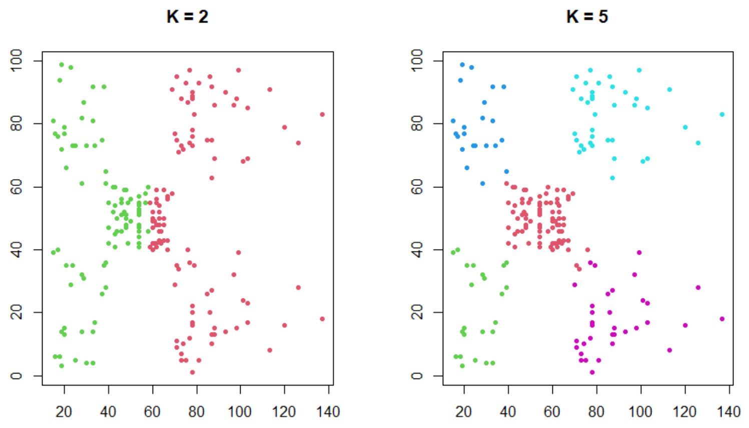


Figure 1: Example of a dataset that has been clustered with K = 2 and K = 5

Algorithm 1: Lloyd's Algorithm [2]

```
1. Choose the number of clusters K;
2. Choose an appropriate initialisation technique and assign centroids using this
   method;
while clusters keep changing do
    a) Allocate each observation to the cluster with the nearest centroid (computed
       with euclidian distance);
    b) Recompute the cluster centroid for each of the K clusters by finding the
       mean vector of the $p$ features for all observations in a cluster;
end
```

This is a batch (offline) algorithm where the transformative step is applied to all observations at once. This makes this algorithm better suited to large datasets than the other algorithms used for K-Means since it's more computationally efficient.

Algorithm 2: MacQueen's Algorithm [1]

```
1. Choose the number of clusters K;
2. Choose an appropriate initialisation technique and assign centroids using this
   method;
while cluster assignments keep changing do
    for observation $i \in (1, ..., n)$ do
        a) Allocate each observation to the cluster with the nearest centroid
           (computed with euclidian distance);
        b) Recompute centroids for the two clusters involved (if observation
           changed cluster);
    end
    Recompute centroids;
end
```

This is an iterative (online) algorithm since the centroids are recalculated every time an observation changes cluster. This can be computationally expensive as n gets large, however, for small n it is efficient since it usually only needs one pass through all observations in order to converge. [4]

Algorithm 3: Hartigan & Wong's Algorithm [19]

```
1. Choose the number of clusters K;
2. Choose an appropriate initialisation technique and assign centroids using this
   method;
3. Allocate observations to their closest cluster centers using Euclidian Distance;
4. Recalculate centroids;
for $j \in (1, ..., k)$ clusters do
    a) Calculate WCSS for cluster $j$;
    for each observation in cluster $j$ do
        i. Calculate the WCSS for each cluster $c$ != $j$ if observation $i$ were to be
           included in that cluster;
        ii. If the WCSS for cluster $c <$ WCSS for cluster $j$, reassign the observation
            to cluster $c$ (where the $WCSS_c = \frac{N_c \sum_i ||x_{ci} - c_c||^2}{N_c - 1}$ for cluster $c$ in this case);
    end
end
```

This algorithm converges very quickly for small n but since it is an iterative algorithm it requires a lot of storage and therefore is computationally expensive for large datasets. [4] Due to the fact these algorithms output a local optimal solution, repeat the algorithms many times with different initial centroids and take the cluster assignment with the smallest total within-cluster sum of squares.

4.1.1 Standardisation

The algorithms described above require distances to be calculated between points. If the variables are on widely different scales, it can seriously impact the resulting clustering. Therefore, when performing clustering analysis on data with variables that have widely different means and/or standard deviations, it is common practice to standardise the data, rescaling all variables to have mean 0 and standard deviation 1. Note: this also applies when performing hierarchical clustering described later.

4.2 Finding The Right Clustering

The algorithms described above require the number of clusters K and an appropriate initialisation method to be specified prior to being performed. There are a number of different methods to choose from to find K and initialise the centroids, and they both have a significant impact on the final clustering, so careful consideration is needed.

4.2.1 Choosing K

Choosing the optimal number of clusters K is one of the most difficult aspects of K-means clustering. Researchers have provided a number of different ways to do this. In figure 1, the dataset has 5 natural clusters and we need a method which would confirm that. Here is a brief summary of some of the methods:

1. **Calinski–Harabasz Index:**[7] Involves maximising the following function

$$CH(K) = \frac{\frac{B(K)}{(K-1)}}{\frac{W(K)}{(N-K)}}$$

   where B(K) is the between cluster sum of squares and W(K) is the within cluster sum of squares, N is the number of observations and K is the number of clusters. This method is a ratio of between-cluster dispersion and within-cluster dispersion. Therefore it makes sense to want to maximise this ratio since K-means is about minimising the variation within clusters while making the clusters as different as possible.

2. **Akaike Information Criterion**[9]**:** The AIC is a way of measuring the quality of a statistical model, taking into account both goodness of fit and parsimony. Here it has been adapted for use with K-means clustering.

$$K = \underset{K}{\operatorname{argmin}}[SSE(K) + 2MK]$$

   where SSE(K) is the sum of squared error for the clustering configuration and M is the dimensionality of the dataset. The aim is to find the value of K which minimises the AIC statistic.

3. **Silhouette Coefficient**[8]**:** This method involves finding the average distance from datapoint x_i to all the other datapoints in the cluster, refered to as $a(i)$, and then the average distance from datapoint x_i to all the datapoints not in the same cluster as x_i, refered to as $b(i)$. Then a silhouette value is calculated for datapoint i by

$$s(i) = \frac{b(i) - a(i)}{max[a(i), b(i)]}$$

   A value of $s(i)$ near to 1 means the datapoint is well-matched to its cluster. A value close to -1 is not very well matched. A value near to 0 means it could be on the border with neighbouring clusters. The mean of the silhouette values over all of the dataset is then taken to be the *silhouette coefficient*. This value is a measure of the quality of the cluster assignment. The way to utilise this method to find the optimal value of K is to find the average silhouette coefficient for different values of K and choosing K that has the highest average.

4. **Elbow Method**[18]: This is a rather different approach to estimating the ideal K since it requires using the observer's judgement instead of relying on a numerical indicator. The method involves creating clusters for a number of different K on the dataset of interest, then plotting K against the total-within-cluster-sum-of-squares. The observer then chooses at what value of K the line starts to flatten out (i.e. the elbow). The general idea is that, as more clusters are added, the reduction in within-cluster sum of squares isn't significant beyond the elbow of the curve. In figure 2 below, there is a clear flattening of the curve at around K = 5. Therefore, K = 5 would be taken to be an estimate of K for this data. Since this is from the same dataset as figure 1, which clearly shows 5 natural clusters, this is a good estimate.

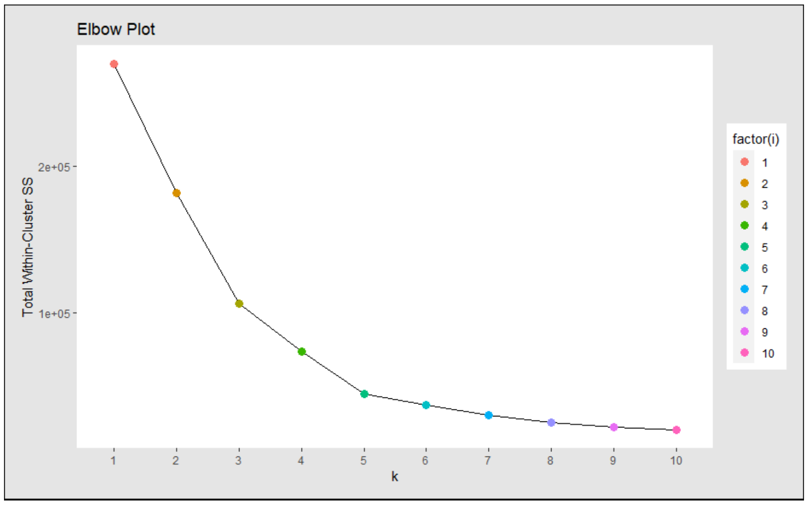


Figure 2: Example of an elbow plot with a clear elbow at K=5

4.2.2 Initialisation Methods

The algorithms described earlier need an initialisation technique in order to have a set of initial centroids to use at the first iteration. There are a range of different methods that vary in complexity. The simplest are random partitioning and the Forgy method but these often lead to bad convergences far away from the optimal clustering. Other methods were made in response to this problem which provide much better initial centroids to help the algorithm to converge to much better clusters.

1. **Random Partitioning**: The simplest initialisation where the n data points are randomly assigned to the k clusters and the intiial centroids are calculated based on this assignment. Since the initial clustering is assigned randomly, the algorithm can often converge to a local optimum value that is far away from the global optimum value. To work around this, it is common practice to repeat the algorithm with different initial random seeds and take the clustering that converges to the lowest objective function value.

2. **Forgy Method**[15]: A simple approach to initialisation in that K random points are chosen as the initial centroids and the data points are then assigned to the centroid that is closest to them. The K-Means algorithm then proceeds with this initial cluster assignment. However, there are drawbacks to this method. The Forgy Method can result in initialisations where some clusters are empty, or where some clusters of data points contain two or more centroids. In both cases, the K-means algorithm will not perform well. In order to overcome this, perform K-means from many different initial configurations, much like with the Random Partitions initialisation.
   Compared to the random partition method, the initial points found by the Forgy method tend to be more spread out.[17]

3. **Hierarchical Clustering**[16]: This technique involves performing hierarchical clustering with Ward's Method on the data or a subsample of the data. Hierarchical clustering is described in more detail in a later section but the way it is utilised for K-Means is as follows:

   (a) Apply hierarchical clustering with Ward's method to cluster the data or a subsample of the data.

   (b) Use the means of the K clusters as the initial centroids for the K-means algorithm.

   Since Ward's method uses the same objective function as K-means, it is the most preferable method.

4. **K-means++**[13]: The standard method of finding the initial cluster centroids can result in bad approximations to the optimal clustering. The K-means++ method

works around this using an algorithmic approach. It does this by first choosing the initial centroid at random, and then choosing the second one to be as far as possible from the first. This is repeated until K centroids have been chosen. In algorithmic form it is this:

   (a) Choose the first centroid at random from the data points.
   (b) Compute the distance $D(x)$ from point x not already chosen as a centroid and the nearest centroid.
   (c) Choose a data point to be a new cluster centre based on the weighted probability distribuition where the probability of choosing a point x is proportional to $D(x)$.
   (d) Repeat steps b and c until K centroids have been found, then proceed with the K-means clustering algorithm described earlier.

5. **Bradley and Fayyad**[14] : This method seeks to find an improved set of initial centroids through a process of resampling and refinement. It chooses j subsamples from the initial data, and performs K-Means clustering with random-partitions on these samples. This produces a set of cluster means. Perform K-means again, this time on the set of centroids produced by all subsamples, with the ith centroid means as the initial starting points. Do this for all j subsamples and take the centroids that minimise the sum of squared error to be the best initialisation. More formally, the method is this:

   (a) For i in $1, ..., j$:
      i. Take a sample from the data, S_i.
      ii. Perform K-means clustering on this sample. If there are any empty clusters, re-center and re-perform K-means on the sample.
      iii. Add the resulting centroids CM_i to a set CM.
   (b) For i in $1, ..., j$:
      i. Perform K-means on the set CM, using CM_i as the initial centroids.
      ii. Add the resulting centroids FM_i to a set FM.
   (c) Choose the final centroids to be the set FM_i which minimises the sum of squared distance between the data points in CM and the closest mean in FM_i.

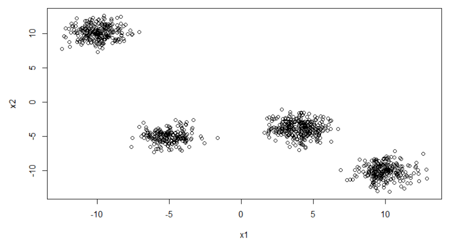
4.3 Applications

K-means clustering can be applied to datasets that are numeric and continuous. The reason K-means clustering is still popular today is due to its ease of implementation, computational efficiency and low memory consumption.[4] MacQueen, the originator of K-means clustering, intended the main use of this technique to be a way for researchers to gain more insights into large multivariable datasets.[1] However, it has been found to be of use in other applications too. Another is to reduce the complexity of large datasets. This has been exemplified in the case of letter grades[5] where numerical grades are clustered into letters and then are represented by the average of each class. Lastly, K-means clustering can be used as an initialization step for more computationally demanding algorithms. By reducing the noise in the data and seperating it, this simplified form of the original data will then be easier for the more demanding algorithms to work with.[6][4]

4.4 Analysis

This section will contain an analysis of coding for K-Means clustering on a constructed dataset. The coding for this analysis is contained in the appendix at the end of the report. To begin with, the dataset is built and the resulting datapoints are distributed as such:

Figure 3: A dataset produced in R with 4 natural clusters



As Figure !! shows, there are obviously four natural clusters in the dataset. However, it is not always obvious how many clusters a dataset contains, in which case, performing one of the techniques discussed earlier for finding K should be used, e.g. average silhouette. The R package used for this analysis is $ClusterR$[23]. It contains a function Optimal_Clusters_KMeans() which calculates the criterion for determining K, for different values of K. For example, it calculates the average silhouette for different values of K and then presents them in a line graph. By performing the function on the dataset, the following graph is obtained.

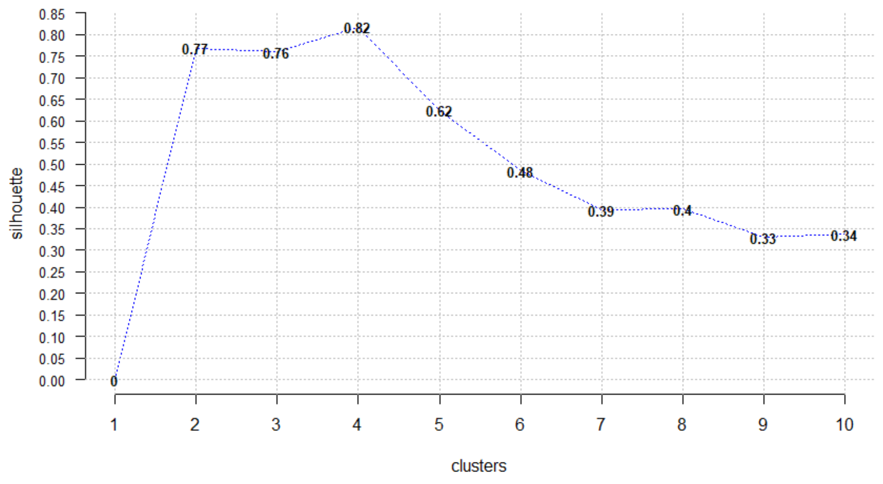


Figure 4: Plot of Average Silhouette over number of clusters K

Other criterion like the BIC and AIC can be specified. The plot in Figure !! is the output of this function and gives the average silhouette for K up to a pre-specificed maximum, in this case 10. For this dataset, the average silhouette coefficient suggests that the optimal value of K is 4 for this dataset, which was expected. The function also takes a initializer argument which tells the function how to determine the inital centroids. In this case, kmeans++ is used.
Another method of determining the optimal value of K is to use the elbow method. As explained earlier, using the observer's judgement of where the curve starts to flatten is taken to be the optimal value of K. The package $factoextra$[24] contains the function fviz_nbclust() which provides a quick and efficient way to plot the elbow curve.

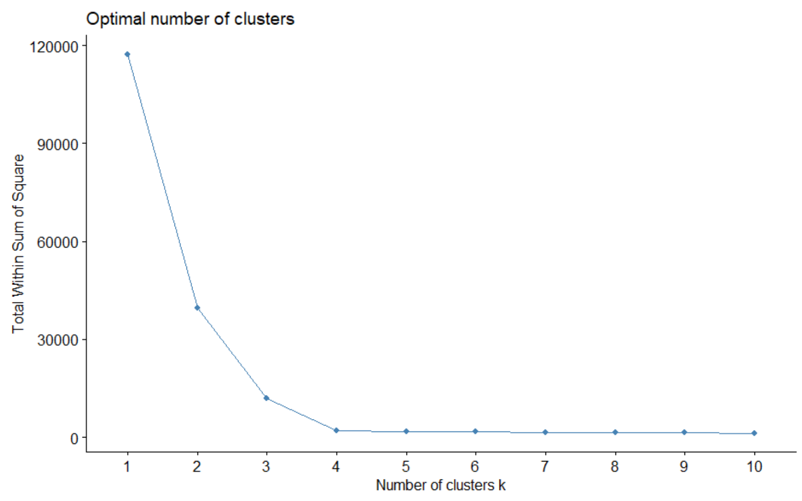


Figure 5: Elbow Curve showing K=4 to be the elbow.

The plot in Figure !! shows the elbow curve for this dataset. It shows a clear visible elbow at K = 4, again indicating that the optimal number of clusters in the dataset is 4. Next, the data will be clustered with the function $KMeans\_rcpp()$ from the $ClusterR$ package. This function allows the kmeans++ initialisation method to be used.

x1	x2
9.954938	-10.085810
-10.058137	9.984557
4.006820	-3.972697
-5.077919	-5.028551

Table 1: Centroids from the resulting clustering

The function gives each data point in the dataset a label of the cluster that it has been allocated to. Here the centroids shown in Table 1 are close to the means of the normal distributions that generated them, showing they have converged close to the real cluster centers.
The function $plot\_2d()$ produces a compact plot of the data and the clusters. It requires the dataset to be 2-dimensional, so principal component analysis can be used in conjunction with it in order to represent the data in two dimensions.

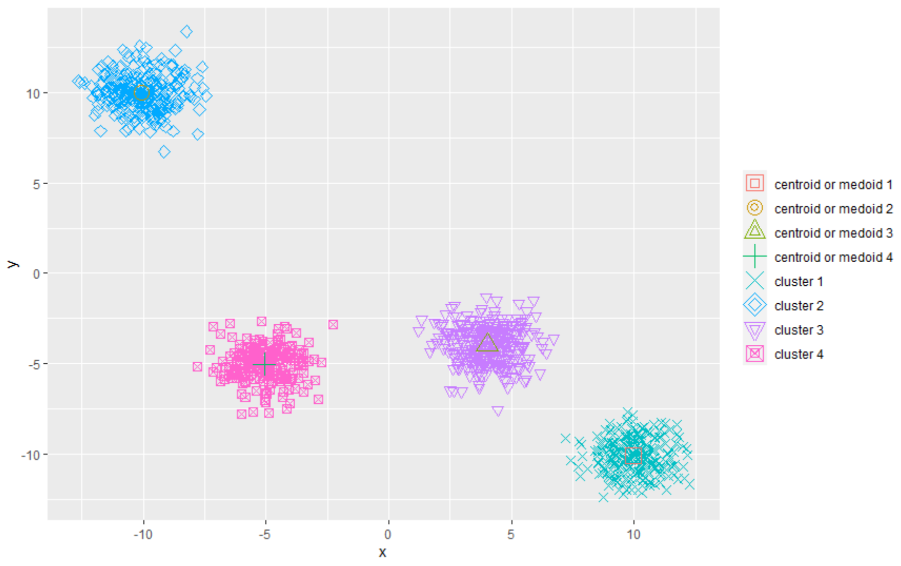


Figure 6: Plot of the data sorted into K=4 clusters, with the centroids also labelled.

For using the other algorithms listed earlier to perform K-means clustering, the standard Kmeans() function in base R provides an option to specify the particular algorithm. It uses the Hartigan-Wong[19] as the default if none is specified.

5 Fuzzy Clustering

This method is a soft version of K-means clustering used when the datasets are complex and the clusters are not clearly defined and may be overlapping. It is soft in the sense that data points belong to a number of clusters simultaneously, whereas in K-means, data points were assigned to single clusters. In this method, each data point is associated with a vector containing the membership to each cluster. Membership can be thought of as the probability of the data point belonging to that specific cluster. Therefore, each element of the vector is between 0 and 1, and the sum of the memberships in each vector is equal to 1. Each membership vector has length equal to a pre-specified integer C, similar to the K in K-means clustering. As a general rule, membership increases the closer a data point is to the center of the cluster.
There exist different algorithms to perform fuzzy clustering, including the Fuzzy C-Means and the Gustafson & Kessel Algorithms. The algorithm mentioned below is the Fuzzy C-Means, since it is the most widely used.

5.1 Fuzzy C-Means Algorithm

This algorithm seeks to minimise the sum of the euclidean distance between all points and cluster centers, where the distance is weighted by the membership of the point to that cluster, i.e. $T = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^{m} ||x_i - c_j||^2$ where u_{ij} is the i, jth membership calculated below, x_i is the ith data point, c_j is the jth cluster centre, and $m \in (1, \infty)$ is the Fuzziness index, a pre-specified parameter that determines how soft the algorithm should be. The algorithm follows these steps:

Algorithm 4: Fuzzy C-Means [10, 11]

```
1. Randomly assign C cluster centers, where C has been previously chosen;
while $\max_{i,j}[|u_{ij}(k+1) - u_{ij}(k)|] < \epsilon$ where $\epsilon$ is $\in (0,1)$ and $k$ is the current iteration do
    for each data point $i$ in the dataset do
        Calculate the membership to each cluster $j$ using $u_{ij} = 1/\sum_{k=1}^{c}(\frac{d_{ij}}{d_{jk}})^{\frac{2}{m}-1}$ where $d_{ij} = \sum_{j=1}^{C} u_{ij}^{m} ||x_i - c_j||^2$ is the euclidian distance between the $ith$ data point and the $jth$ cluster centre;
    end
    Compute the $jth$ cluster center by $v_j = \frac{\sum_{i=1}^{n} u_{ij}^{m} x_i}{\sum_{i=1}^{n} u_{ij}^{m}}$ for all $c$ clusters;
end
```

Similar to the K-means clustering before, there are different methods available for determining the ideal number of clusters c. Commonly, the average silhouette is calculated for different values of c and the value of c that provides the maximum average is considered the ideal value.[8]

5.2 Analysis

In this section, a fuzzy clustering analysis will be performed on the same constructed dataset as in the previous analysis, as well as a more noisy dataset in order to contrast the differences between fuzzy clustering on obviously grouped data and less obvious data. The package $fclust$[25] is used since it contains functions specifically aimed at fuzzy clustering. Refer to the appendices for the code used in this analysis.
By performing a silhouette plot, the ideal number of clusters can again be obtained. The functions $FKM()$ and $SIL()$ are used, the first of which performs fuzzy C-means using the FCM algorithm, and the second of which calculates the average silhouette for that clustering.

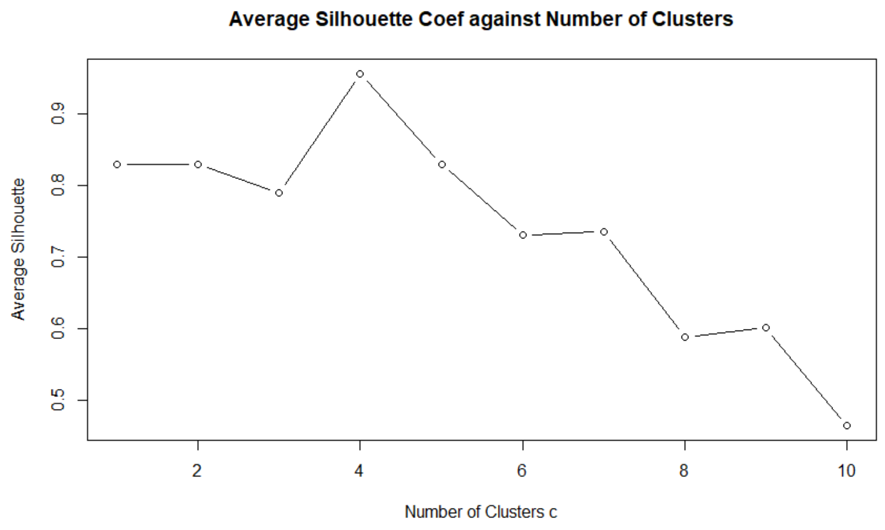


Figure 7: Plot of average silhouette against number of clusters.

The plot in Figure !! shows that the optimal value of K is 4, according to the silhouette coefficient.
Next, the data can then be clustered with the ideal value of c and the results can be analysed to see if this clustering is a good fit. The function *plot.fclust()* is used to plot the results.

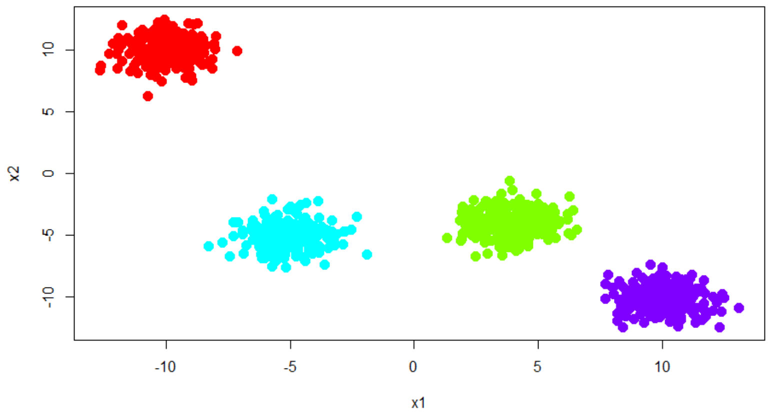


Figure 8: Plot showing the fuzzy clustering

Figure !! shows how, similar to how K-means performed, the dataset was grouped into 4 clusters. Since the dataset contains four naturally distinct groups of data, the fuzzy clustering performs a similar function to the K-means function. The function $VIFCR()$ then allows the analyst to inspect the results and determine if the clustering fits the data well. The following three plots are produced:

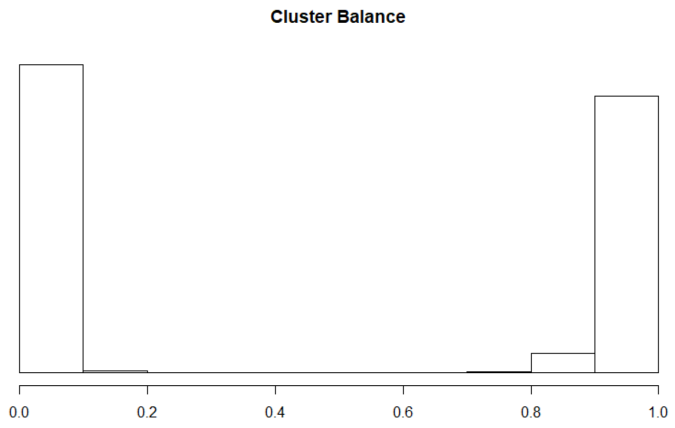


Figure 9: Histogram of memberships, where the memberships are grouped from 0 to 1 by 0.1

The plot in Figure !! is a histogram of the memberships. Ideally, an optimal clustering will have high rectangles at the beginning and end, with small ones in the middle, since this would indicate that the data points have non-ambiguous cluster memberships.

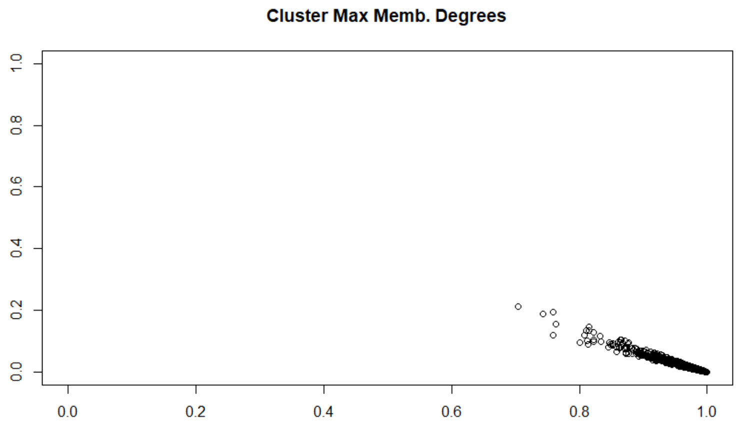


Figure 10: Scatterplot of the two highest memberships for each datapoint, (u_1, u_2)

The plot in Figure !! is a scatterplot for the two highest memberships of a datapoint, with the highest on the x-axis and second on the y-axis. The datapoints can only occupy the inside of the triangle made by the points (0,0), (0.5,0.5), and (1,0). A clear clustering will have points close to (1,0). If points are closer to (0.5,0.5), this indicates these points have ambiguous memberships to two clusters. Points near (0,0) have low membership to all clusters and so may be considered outliers in the data.

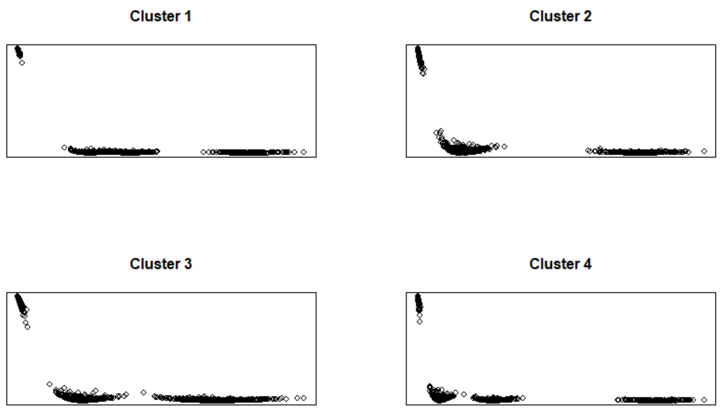


Figure 11: Scatterplots of distance from cluster center against membership of cluster

The last plot in Figure !! shows the distances of each point from the cluster center

against the membership to that cluster. Ideally, points will be in the upper left and lower right since this indicates high memberships for small distances and low memberships for large distances. Points away from this indicate, say in the bottom left, indicate datapoints with ambiguous membership assignments for two or more clusters.
These plots show that the clustering fits very well for this dataset and that there is very little ambiguity about cluster membership. Next, the same procedure will be performed on a noisier dataset in order to demonstrate how the above plots would appear when dealing with more ambiguous cluster assignments.

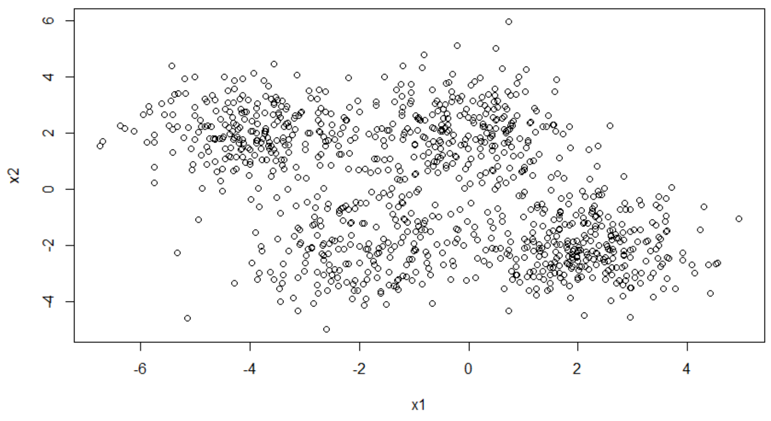


Figure 12: Scatterplot of the noisier dataset

As the plot in Figure !! shows, this dataset has less clearly defined clusters where many datapoints could potentially belong to more than one cluster. The following plots result from performing fuzzy clustering on this dataset followed by the function $VIFCR()$ which produces the same plots as mentioned earlier.

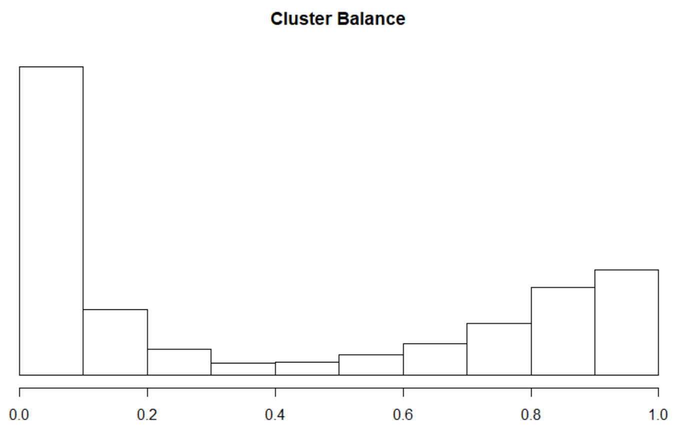


Figure 13: Histogram of memberships for noisy data, where the memberships are grouped from 0 to 1 in intervals of 0.1

The histogram in Figure 14 shows the distribution of memberships. Compared to Figure 10, the rectangles are less well balanced with many more memberships between 0 and 0.1, and fewer between 0.9 and 1. There are also more memberships between 0.2 and 0.8 than in Figure 10, showing less certainty of belonging to a single cluster.

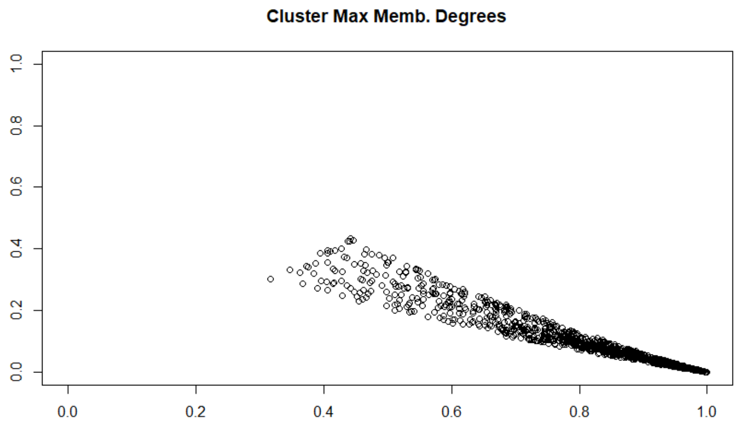


Figure 14: Scatterplot of the two highest memberships for each datapoint in noisy data, (u_1, u_2)

Compared to the same plot for the less noisy data, the scatterplot in Figure !! shows how the highest memberships for datapoints in the noisier data are more dispersed, i.e much less than one. Again, this highlights the ambiguity of a lot of these datapoints as to which cluster they belong to. The previous plot for much cleaner data in Figure 11 had all the points congregating towards the lower right hand corner, since its highest cluster memberships were much closer to one.

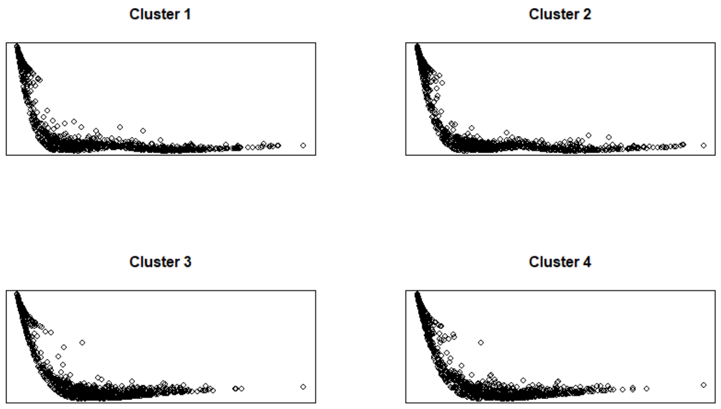


Figure 15: Scatterplots of distance (x-axis) from cluster center against membership of cluster (y-axis) for X_noisy

For an ideal clustering, the resulting plots should contain clear seperations of the datapoints into their natural clusters, as shown in Figure !!!!. Datapoints close to a cluster center should have a high degree of membership to that cluster. Points further from the cluster centre should have a lower degree of membership. However, in Figure !!, the datapoints are not seperated and there are a lot of points in the lower left, again indicating less well defined membership to said clusters. These datapoints, for whom membership to a cluster is not clear, have the opportunity to belong to more than one cluster based on their first and second highest memberships.

6 Hierarchical Clustering

This method does not require you to prespecify the number of clusters, because the output can be split up into any number of clusters (up to a maximum of n) based on how the output is divided. The output of hierarchical clustering is a dendrogram, as seen in Figure 12, which is a tree-like diagram that contains all the observations along the bottom and branches connecting them together based on similarity above. In interpreting a dendrogram, the similarity of observations is given by the height that fusion between branches/observations occurs on the vertical axis, i.e the more similar the observations, the lower down the dendrogram their branches fuse.

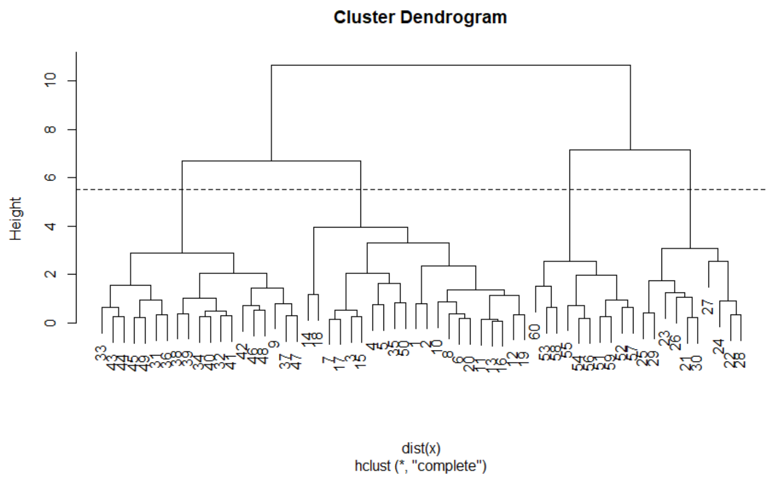


Figure 16: Example of a dendrogram from hierarchical clustering performed with complete linkage on a practice dataset, with a horizontal cut defining four clusters underneath.

The resulting dendrogram can then be cut by a horizontal line and the resulting fused branches below the cut can be considered whole clusters. If you want more clusters, cut the dendrogram at a lower place.
There are two approaches to hierarchical clustering: agglomerative and divisive. Agglomerative clustering starts with single clusters (containing only one data point) and joins them together based on how similar they are, until they are all contained in one cluster. Divisive clustering starts with all data points in one cluster and divides them based on how dissimilar they are, until all the data points are seperated into individual clusters.

6.1 Agglomerative Hierarchical Clustering Algorithm

When performing hierarchical clustering, a type of linkage needs to be specified. Linkage methods are how the dissimilarity between two clusters is calculated. This also requires a dissimilarity measure to be chosen. This is how the distance between data points is calculated. Then the algorithm proceeds by starting with all observations as individual clusters. Using the choice of dissimilarity measure, the clusters that are most similar are fused together into one cluster. Checking similarity again, this time with $n-1$ clusters, the most similar are joined together. This process continues until all observations belong to a single cluster. More formally, the algorithm is this:

Algorithm 5: Agglomerative Hierarchical Clustering Algorithm [3]

```
1. Starting with all $n$ observations and a choice of dissimilarity measure and
   linkage, find the $\binom{n}{k}$ dissimilarities between all observations (treating the
   observations as individual clusters);
repeat
    a) Fuse together the two clusters with the lowest dissimilarity. The height of
       the fusion on the dendrogram is the dissimilarity;
    b) Calculate the pairwise inter-cluster dissimilarities for the new set of clusters;
until all observations contained in one cluster;
```

6.1.1 Linkage Methods

The algorithm requires us to calculate the dissimilarity between clusters where one or both of the clusters can be groups of observations. This is done using linkage methods. The following is a list of some of the different types of linkage:[21]

- **Complete**: Using a choice of dissimilarity measure (e.g Euclidian Distance), all pairwise dissimilarities between all obervations in cluster A and all observations in cluser B are calculated. Then the largest of these is chosen to be the dissimilarity measure for this pair. This method allows for finding clusterings in compact data, however it is sensitive to outliers.

- **Single**: Same as complete but the smallest of the pairwise dissimilarities is chosen instead of the largest. The advantages of this method are that it can recognise elliptical and elongated shapes in the data, where the other methods described fail to do so. However, it responds strongly to outliers, sometimes resulting in bad clusterings.

- **Average**: The pairwise dissimilarites between all observations in cluster A and cluster B are calculated, in the same way as complete and single. Then the average of all of them is used as the dissimilarity measure for this pair.

- **Centroid**: The centroid (a vector of the mean values of each feature) for cluster A and cluster B is calculated. Then the dissimilarity between the two centroids is found using the choice of dissimilarity measure and taken to be the dissimilarity for this pair.

- **Ward's Method**[3]: This provides a unique way of measuring how similar two clusters are. It says that the distance between two clusters is the amount by which the within-cluster sum of squares increases when the two clusters are merged. Since the within-cluster sum of squares increases as clusters are merged into bigger clusters, Ward's Method seeks to keep this increase to a minimum by merging the clusters that provide the smallest increase in the within-cluster sum of squares. The formula to calculate this increase when merging clusters i and j is such: $\nabla(i,j) = \frac{n_i n_j}{n_i + n_j}||C_i - C_j||^2$ where n_a is the number of points in a cluster a and C_a is the centroid for a cluster a.

Generally, average and complete linkage are used over single linkage and centroid linkage. This is because single linkage has a tendancy to result in long trailing clusters where single observations are fused at each stage, resulting in a dendrogram that is unbalanced and harder to interpret. A big flaw in using centroid linkage is that it can result in inversions. This is where two clusters are fused below where the individual clusters lie on the dendrogram. This makes visualising and interpreting such a dendrogram more difficult.

6.1.2 Dissimilarity Measures

The most popular choice of dissimilarity measure is Euclidian distance, but depending on the type of data you are clustering and the aim of the data exploration, other choices may be more appropriate. Other examples of dissimilarity measures between observations a and b include: [22]

- **Squared Euclidian Distance**: $\|a-b\|_2^2 = \sum_{i=1}^{n}(a_i - b_i)^2$
- **Manhattan Distance**: $\|a-b\|_1 = \sum_{i=1}^{n}|a_i - b_i|$. Used over euclidian distance when the data is of high dimensionality.[20]
- **Maximum Distance**: $\|a-b\|_\infty = \max_i |a_i - b_i|$

6.2 Divisive Hierarchical Clustering Algorithm

Another way to perform hierarchical clustering is via a divisive algorithm, where the points all start in one cluster and are gradually split until each point is contained in its own cluster. The standard way it is performed is as such:

Algorithm 6: Divisive Hierarchical Clustering Algorithm [8]

```
1. Start with all points contained in one cluster;
repeat
    a) Split the cluster into two based on maximising the reduction in the sum of
    squared error after splitting the cluster;
    b) Choose the cluster with the highest within-cluster sum of squared error;
until all points are contained in individual clusters;
```

Part a) of the iteration is difficult to perform with large n since there are $2^{n-1} - 1$ ways of splitting the data. Therefore methods are used to avoid having to perform such large numbers of calculations. One way is to use Bisecting K-Means to maximise Ward's Distance for all cluster combinations. [12]

6.3 Evaluation of Clusters

In unsupervised learning, it is not as clear and straightforward to determine how well the clustering has been performed, since there is no test data available. Therefore, certain internal evaluation techniques have been created to give an estimation of how good a clustering is. However, these internal methods only really allow comparisons of clusterings for the same dataset. The silhouette coefficient described in the K-means section can be used as an evaluation measure. Another measure is the Dunn Index.

6.3.1 Dunn Index

This is the ratio between the minimum inter-cluster distance and the maximal intra-cluster distance: $DI = \frac{\min_{1<i<j<n} d(i,j)}{\max_{1<k<n} d'(k)}$ where $d(i,j)$ is the distance between clusters i and j, and $d'(k)$ is the intra-cluster distance for cluster k. $d'(k)$ can be calculated using a number of ways, such as maximal distance between points in cluster K. A higher Dunn Index indicates a better clustering.[10]

6.4 Usage

There are a number of disadvantages for hierarchical clustering. Firstly, when dendrograms are dense with large n, it is not obvious where to cut the dendrogram, and so they become very uninterpretable. Also, the hierarchical nature of the dendrogram means that clusters are nested, which is not always optimal. In reality, often clusters are not nested, and these clusters will not be observed according to this technique. Therefore more often than not hierarchical clustering will lead to inferior results than other clustering techniques for the same number of clusters. However, due to the flexability in the number of clusters, and the variety of parameters available to tweek, it is one of the most popular clustering methods to use.
I plan to include some applcations of the hierarchical clustering method.

7 Appendix: Code

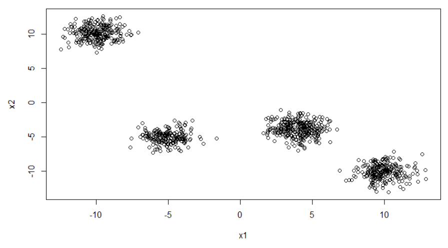
7.1 K-Means Clustering

In this section, a clustering analysis will be performed in R on an improvised dataset made to have 4 natural clusters. The R package *ClusterR* [23] will be used.

```
N <- 1000
x <- matrix(0, N, ncol=2)
x[1:300,1]<- rnorm(300, mean=4, sd=1)
x[1:300,2]<-rnorm(300, mean=-4, sd=1)
x[301:500,1] <- rnorm(200, mean=-5, sd=1)
x[301:500,2] <- rnorm(200, mean=-5, sd=1)
```

```
x[501:750, 1]<-rnorm(250, 10, 1)
x[501:750,2] <- rnorm(250, mean=-10, sd=1)
x[751:1000,1]<-rnorm(250, mean=-10,sd=1)
x[751:1000,2]<-rnorm(250,mean=10,sd=1)
x<- as.data.frame(x)
colnames(x) <- c("x1","x2")
```

Figure 17: A dataset produced in R with 4 natural clusters



This gives the dataset in Figure 3. In an analysis where the natural number of clusters is not known, performing one of the techniques discussed earlier, for example, finding the average silhouette coefficient, in order to find K would be an appropriate next step. Here the function Optimal_Clusters_KMeans() from the package ClusterR is used. This function allows different criteria for selecting K to be given. Options besides "silhouette" include "AIC", "BIC" and "variance explained".

```
library("ClusterR")
opt_kmeans = Optimal_Clusters_KMeans(x, max_clusters = 10, criterion =
    "silhouette", initializer = "kmeans++", num_init = 100)
```

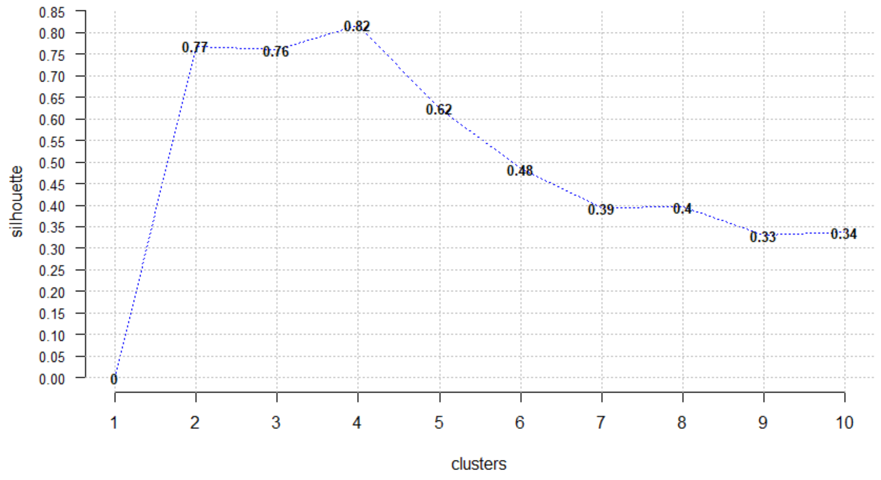


Figure 18: Plot of Average Silhouette over number of clusters K

The plot shown in figure 4 is the output of the function and gives the average silhouette for each K clustering up to the maximum specified in the function-call, in this case 10. The num_init argument tells the function how many times to repeat the algorithm with different initial centroids. The initializer argument determines how the initial centroids are initialised. In this case, kmeans++ is used, since it provides much better starting centroids than random partitioning.
As stated in the previous section, a higher average silhouette indicates that the data points in the cluster fit their assigned clusters better. This plot shows K=4 as having the highest average silhouette, so going forward, we will cluster with K=4. Additionally, the elbow method provides a simple way to determine the optimal number of clusters. As explained earlier, using the observer's judgement of where the curve starts to flatten is taken to be the optimal value of K. The package $factoextra$[24] contains the function fviz_nbclust() which provides a quick and efficient way to plot the elbow curve.

```
library("factoextra")
fviz_nbclust(x = x,kmeans,method = "wss")
```

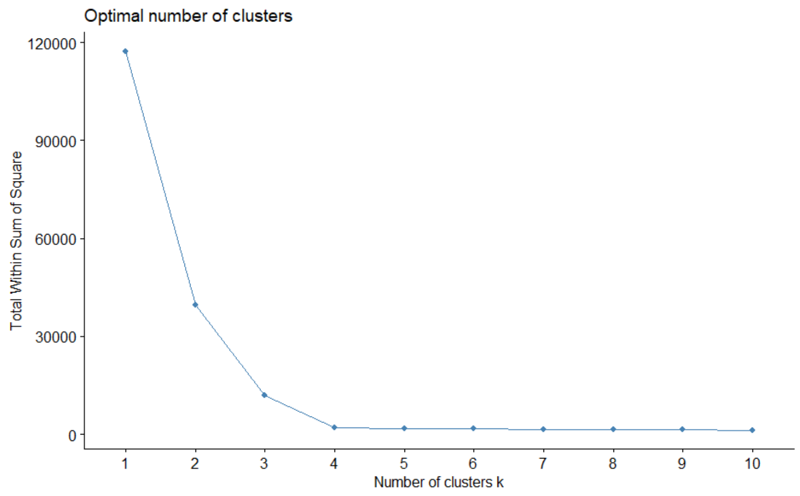


Figure 19: Elbow Curve showing K=4 to be the elbow.

As figure 5 shows, the curve starts to flatten at K=4, agreeing with the silhouette method that the data contains 4 clusters.
Next, the data will be clustered with the function *KMeans_rcpp*() from the *ClusterR* package. This function allows the kmeans++ initialisation method to be used.

```
km = KMeans_rcpp(x, 4, 100, initializer = "kmeans++")
head(km$clusters,n = 10)
 [1] 3 3 3 3 3 3 3 3 3 3
km$centroids
          [,1]       [,2]
[1,]   9.954938 -10.085810
[2,] -10.058137  9.984557
[3,]   4.006820  -3.972697
[4,]  -5.077919  -5.028551
```

The function gives each data point in the dataset a label of the cluster that it has been allocated to. Here the centroids are close to the means of the normal distributions that generated them, showing they have converged close to the real cluster centers.
The function plot_2d() produces a compact plot of the data and the clusters. It requires the dataset to be 2-dimensional, so principal component analysis can be used in conjunction with it in order to represent the data in two dimensions.

```
plot_2d(data=x, clusters=km$clusters, centroids_medoids=km$centroids)
```

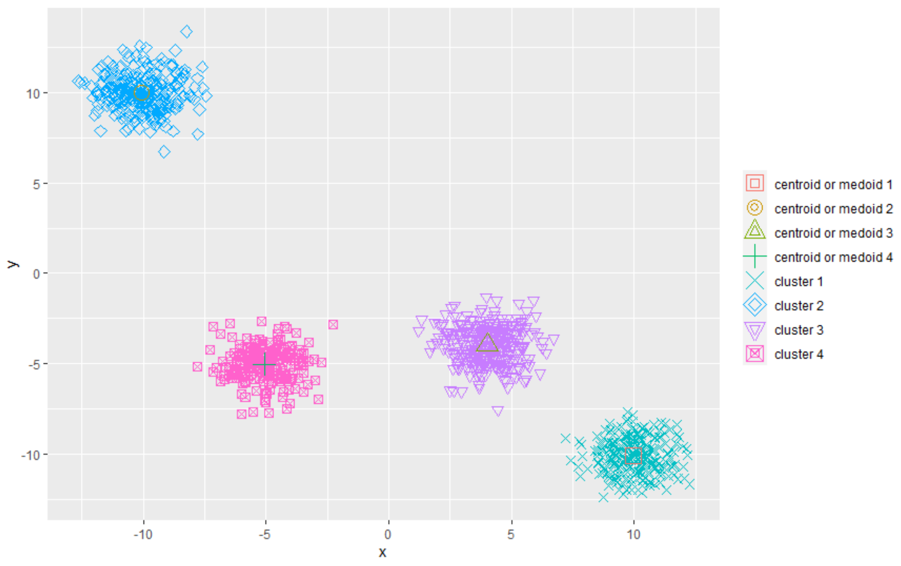


Figure 20: Plot of the data sorted into K=4 clusters, with the centroids also labelled.

For using the other algorithms listed earlier to perform K-means clustering, the standard Kmeans() function in base R provides an option to specify the particular algorithm.

```
kmeans(x, centers, iter.max = 10L, nstart = 1L, algorithm = c("Hartigan-Wong",
    "Lloyd", "Forgy", "MacQueen"), trace = FALSE)
```

It uses the Hartigan-Wong[19] as the default if none is specified.

7.2 Fuzzy Clustering

The dataset from the previous example will be fuzzy-clustered using the functions attached to the package *fclust*. This package contains all the necessary components needed to perform Fuzzy clustering on a dataset.
Firstly, the package *fclust*[25] is activated. Then, the clustering analysis is performed on the data and in the process, the ideal value of C will be determined. By performing the following code, a silhouette plot can be obtained that will determine what C should be for the data. The functions $FKM()$ and $SIL()$ are used, the first of which performs fuzzy C-means using the FCM algorithm, and the second of which calculates the average silhouette for that clustering.

```
library("fclust")
sil_list <- matrix(data = 0,nrow = 10,byrow = TRUE)
for (i in 1:10){
  fuzzc <- FKM(x,k=i)
  sil_ind <- SIL(Xca = x,U = fuzzc$U)
  sil_list[i,]<-sil_ind$sil
  }
plot(y=sil_list, x=1:10,type = "b", main = "Average Silhouette Coef against
    Number of Clusters", xlab = "Number of Clusters c", ylab = "Average
    Silhouette")
```

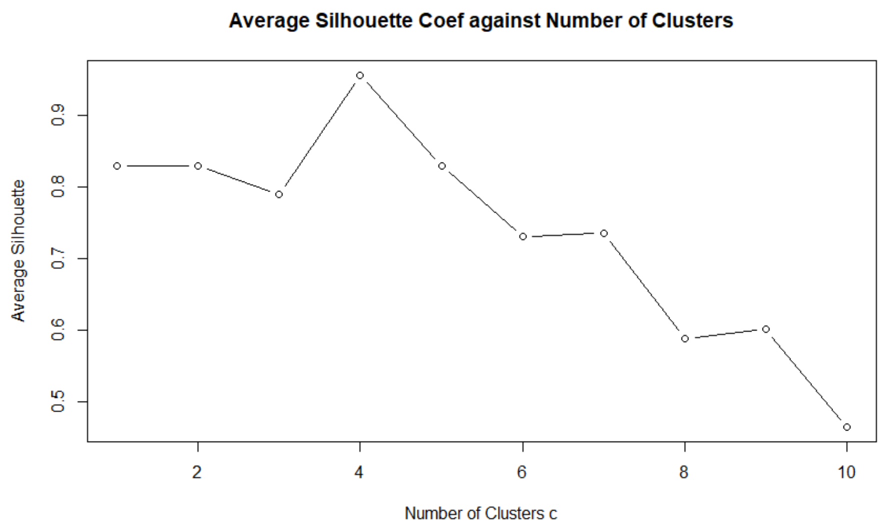


Figure 21: Plot of average silhouette against number of clusters.

The plot in figure 7 shows that the optimal value of K is 4, according to the silhouette coefficient.
Next, the data can then be clustered with the ideal value of c and the results can be analysed to see if this clustering is a good fit. The function *plot.fclust*() is used to plot the results.

```
fuzz <- FKM(x, k=4)
plot.fclust(fuzz)
```

Figure 8 shows how, similar to how K-means performed, the dataset was grouped into 4 clusters. Since the dataset contains four naturally distinct groups of data, the fuzzy clustering performs a similar function to the K-means function. The following function $VIFCR()$ then allows the analyst to inspect the results and determine if the clustering fits the data well.

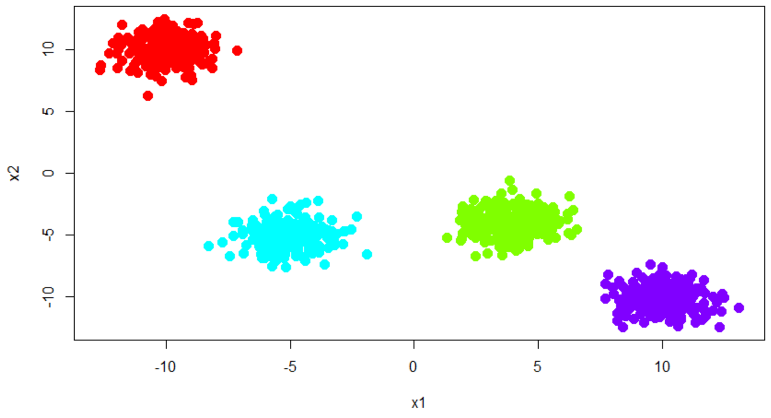


Figure 22: Plot showing the fuzzy clustering

```
VIFCR(fuzz)
```

This code produces three plots that can help to determine the validity of the fuzzy clustering. The plot in Figure 9 is a histogram of the memberships. Ideally, an optimal clustering will have high rectangles at the beginning and end, with small ones in the middle, since this indicates that data points have non-ambiguous cluster memberships.

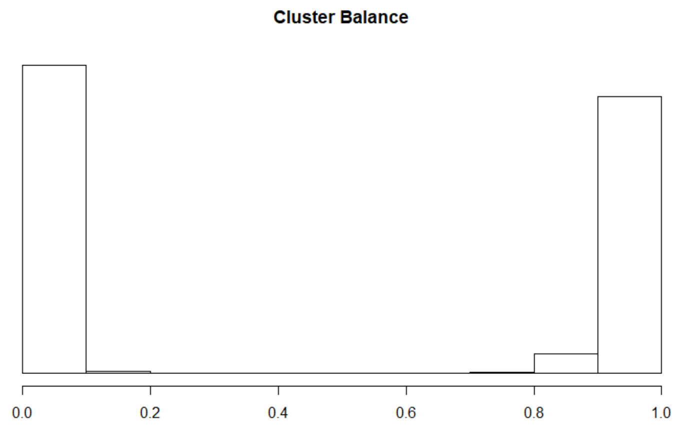


Figure 23: Histogram of memberships, where the memberships are grouped from 0 to 1 by 0.1

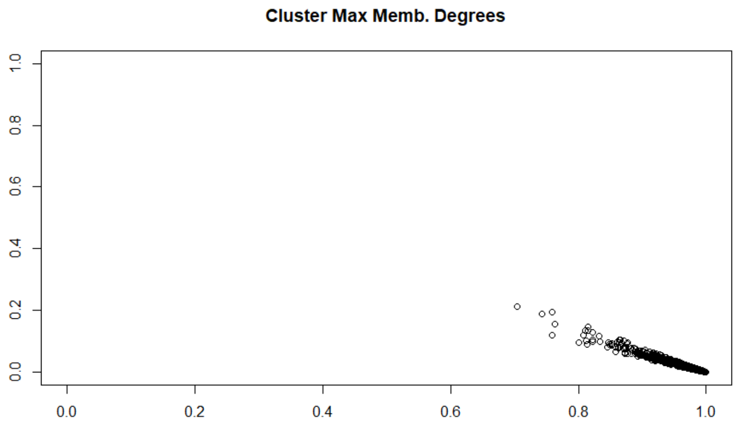


Figure 24: Scatterplot of the two highest memberships for each datapoint, (u_1, u_2)

Figure 10 contains the scatterplot for the two highest memberships of a datapoint, with the highest on the x-axis and second on the y-axis. The datapoints can only occupy the inside of the triangle made by the points (0,0), (0.5,0.5), and (1,0). A good clustering will have points close to (1,0). If points are closer to (0.5,0.5), this indicates these points have ambiguous memberships to two clusters. Points near (0,0) have low membership to all clusters and so may be considered outliers in the data.

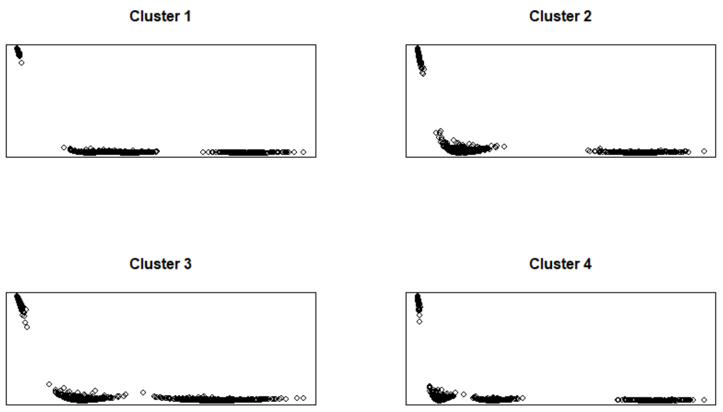


Figure 25: Scatterplots of distance from cluster center against membership of cluster

The last plot in Figure 11 shows the distances of each point from the cluster center

against the membership to that cluster. Ideally, points will be in the upper left and lower right since this indicates high memberships for small distances and low memberships for large distances. Points away from this indicate, say in the bottom left, indicate datapoints with ambiguous membership assignments for two or more clusters.
These plots show that the clustering fits very well for this dataset and that there is very little ambiguity about cluster membership. Next, the same procedure will be performed on a noisier dataset in order to demonstrate how the above plots would appear when dealing with more ambiguous cluster assignments.

```
N <- 1000
X_noisy <- matrix(0, N, ncol=2)
X_noisy[1:300,1]<- rnorm(300, mean=2, sd=1)
X_noisy[1:300,2]<-rnorm(300, mean=-2, sd=1)
X_noisy[301:500,1] <- rnorm(200, mean=-2, sd=1)
X_noisy[301:500,2] <- rnorm(200, mean=-2, sd=1)
X_noisy[501:750, 1]<-rnorm(250, mean = 0, sd=1)
X_noisy[501:750,2] <- rnorm(250, mean=2, sd=1)
X_noisy[751:1000,1]<-rnorm(250, mean=-4,sd=1)
X_noisy[751:1000,2]<-rnorm(250,mean=2,sd=1)
X_noisy <- as.data.frame(X_noisy)
colnames(X_noisy) <- c("x1","x2")
plot(X_noisy)
```

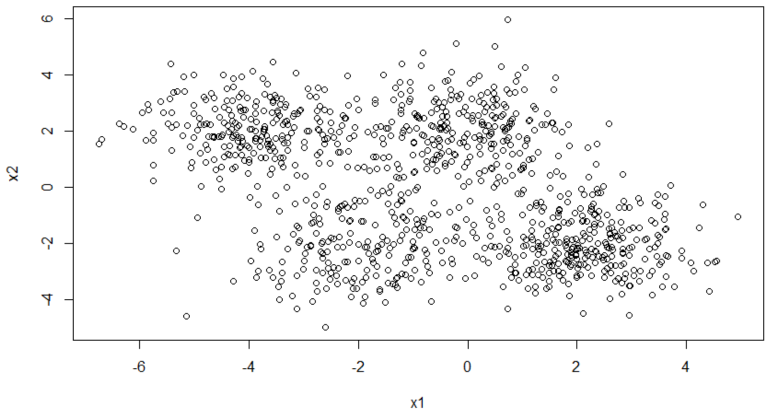


Figure 26: Scatterplot of the dataset X_noisy

As Figure 13 shows, this dataset has less clearly defined clusters where many datapoints could potentially belong to more than one cluster. The following code performs fuzzy clustering on this dataset followed by the function $VIFCR()$ which determinees it's validity through plots as shown earlier.

```
fuzz_noise <- FKM(X_noisy, k=4)
VIFCR(fuzz_noise)
```

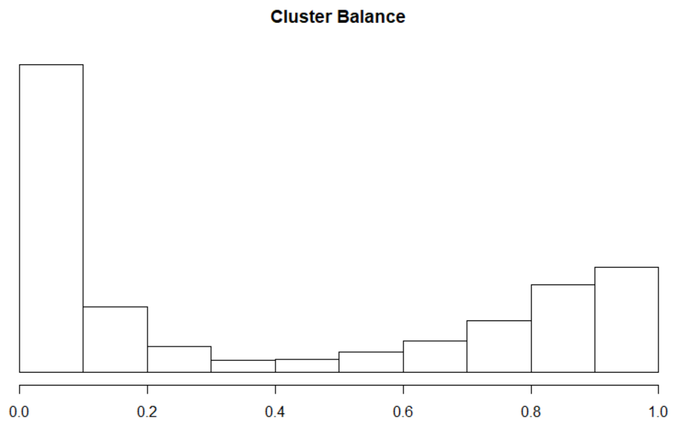


Figure 27: Histogram of memberships for noisy data, where the memberships are grouped from 0 to 1 in intervals of 0.1

The histogram in Figure 14 shows the distribution of memberships. Compared to Figure 10, the rectangles are less well balanced with many more memberships between 0 and 0.1, and fewer between 0.9 and 1. There are also more memberships between 0.2 and 0.8 than in Figure 10, showing less certainty of belonging to a single cluster.

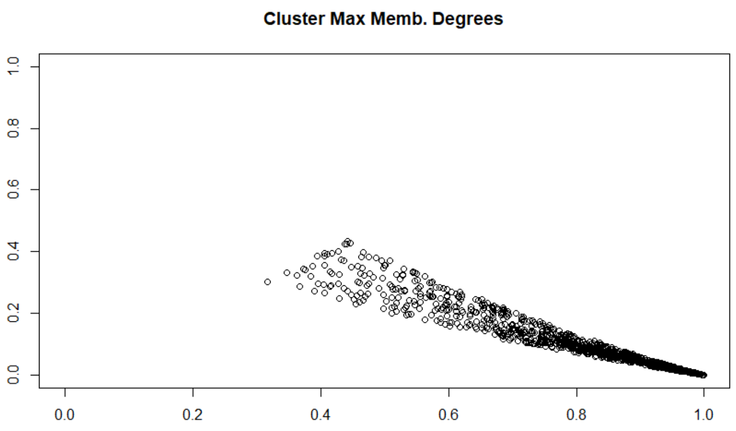


Figure 28: Scatterplot of the two highest memberships for each datapoint in X_noisy, (u_1, u_2)

Compared to Figure 11, the scatterplot in Figure 15 shows how the highest memberships for datapoints in X_noisy (x-axis) are more dispersed, i.e much less than one. Again, this highlights the ambiguity of a lot of these datapoints as to which cluster they belong to. The previous plot for much cleaner data in Figure 11 had all the points congregating towards the lower right hand corner, since its highest cluster memberships were much closer to one.

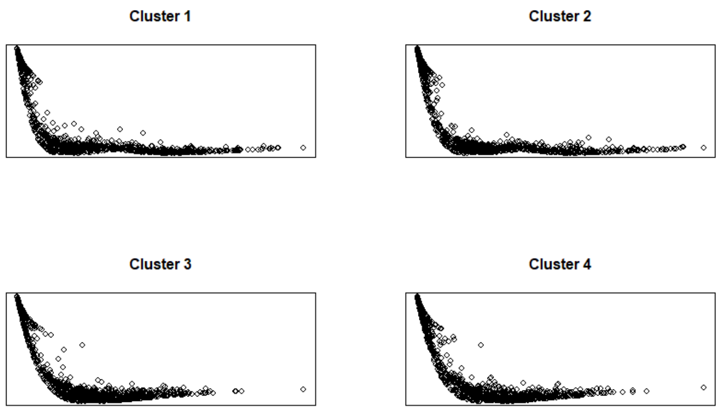


Figure 29: Scatterplots of distance (x-axis) from cluster center against membership of cluster (y-axis) for X_noisy

For an ideal clustering, as shown in Figure 12, the resulting plots should contain clear seperations of the datapoints into their natural clusters. Datapoints close to a cluster center should have a high degree of membership to that cluster. Points further from the cluster centre should have a lower degree of membership. However, in Figure 16, the datapoints are not seperated and there are a lot of points in the lower left, again indicating less well defined membership to said clusters. These datapoints, for whom membership to a cluster is not clear, have the opportunity to belong to more than one cluster based on their first and second highest memberships.

7.3 Hierarchical Clustering

7.3.1 Agglomerative Clustering

In this section, the hierarchical clustering linkage methods specified earlier will be shown in examples. To begin with, the following code produces a dataset that contains two nested ellipses. The resulting ellipses are shown in Figure 13.

```
elliptical <- function(x, a, b){
  y_pos <- sqrt((b^2)*(1-(x/a)^2))
  y_neg <- -sqrt((b^2)*(1-(x/a)^2))
  y <- c(y_pos, y_neg)
  return(y)
  }
x2 <- seq(from=-10, to=10, by=0.5)
y2 <- elliptical(x2, 10, 5)
x2 <- c(x2,x2)
x1 <- seq(from=-5, to=5, by = 0.1)
y1<- elliptical(x1,5,2)
x1 <- c(x1,x1)
x<- c(x1,x2)
y<-c(y1,y2)
# Add a bit of noise to the data
x<-jitter(x, amount = 0.5)
y<-jitter(y, amount = 0.5)
el_data <- cbind(x,y)
plot(el_data)
```

The function *hclust*() from base R is used to perform hierarchical clustering. It takes a distance matrix and a method of linkage as standard arguments. Then the function *cutree*() is used to bisect the dendrogram at a certain level, depending on how many clusters are required. It needs to be provided with the *hclust*() output (a dendrogram) and the number of clusters required.
When the three most common types of linkage are performed on the more complex type of spatial data, the following occurs:

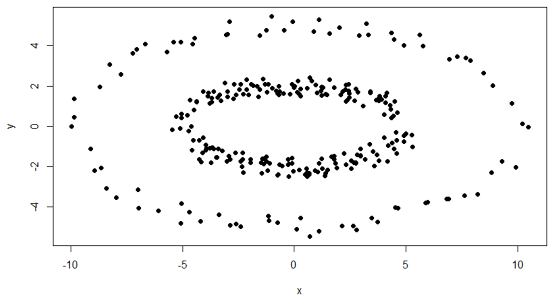


Figure 30: Elliptical data produced by the above code with centers (0,0).

```
par(mfrow=c(1,3))
hc.single <- hclust(dist(el_data),method="single")
plot(el_data, pch=19, col=cutree(hc.single, 2), main = "Single Linkage")
hc.complete <- hclust(dist(el_data), method="complete")
plot(el_data, pch=19, col=cutree(hc.complete, 2), main="Complete Linkage")
hc.average <- hclust(dist(el_data),method = "average")
plot(el_data, pch=19, col=cutree(hc.average, 2), main="Average Linkage")
```

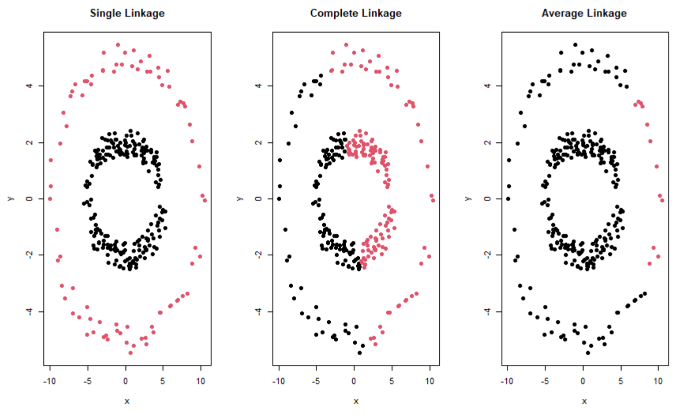


Figure 31: The results of performing hierarchical clustering on the ellipse data with single, complete and average linkage.

As shown in Figure 14, the single linkage captures the elliptical shape of the data while the other types of linkage do not. This is one of the strengths of single linkage since it allows for clustering on data which is more complex in shape. Applying the silhouette coefficient measure to each of the clusterings gives the following results:

```
> mean(silhouette(cutree(hc.single, 4), dist(el_data))[,"sil_width"])
[1] 0.2169047
> mean(silhouette(cutree(hc.average,4),dist(el_data))[,"sil_width"])
[1] 0.06023196
> mean(silhouette(cutree(hc.complete,4),dist(el_data))[,"sil_width"])
[1] 0.2811164
```

This evaluation metric implies that complete linkage may provide the best clustering. However, based on a visual analysis, it is obvious single linkage provides the most appropriate clustering. This shows the evaluation metrics are not a perfect or overly reliable indicator of the quality of the clustering. Their main use is to provide a form of feedback to help move in the right direction.

Single linkage doesn't produce great results on data which is noisy and not well defined, as shown in Figure 15. The following code produces a more noisy dataset and then applies the same procedure to cluster the data according to the different types of linkage.

```
par(mfrow=c(1,3))
x3 <- matrix(0, 600, ncol=2)
x3[1:200,1]<- rnorm(200, mean=1.5, sd=0.85)
x3[1:200,2]<-rnorm(200, mean=-1.5, sd=0.85)
x3[201:300,1] <- rnorm(100, mean=-1.5, sd=0.85)
x3[201:300,2] <- rnorm(100, mean=-1.5, sd=0.85)
x3[301:500, 1]<-rnorm(200, mean = 1.5, sd=0.85)
x3[301:500,2] <- rnorm(200, mean=1.5, sd=0.85)
x3[501:600,1]<-rnorm(100, mean=-1.5,sd=0.85)
x3[501:600,2]<-rnorm(100,mean=1.5,sd=0.85)
x3<- as.data.frame(x3)
colnames(x3) <- c("x1","x2")
hclust.single <- hclust(d = dist(x3),method = "single")
plot(x3, pch=19, col=cutree(hclust.single, k=4))
hclust.complete <- hclust(d=dist(x3), method="complete")
plot(x3, pch=19,col=cutree(hclust.complete, k=4))
```

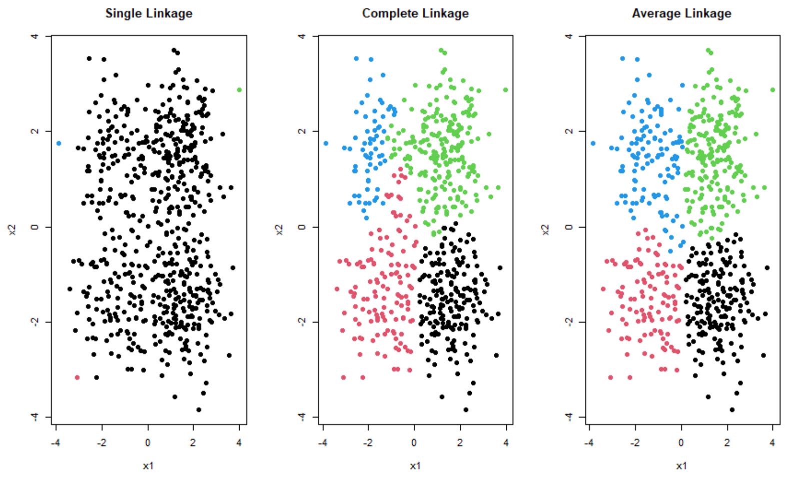


Figure 32: Results of performing the above code on the noisy dataset.

Figure 15 shows the results of the code. The single linkage does not perform nearly as well as average or complete. This is because it is sensitive to outliers in the data and treats them as their own cluster. Therefore it results in extremely unbalanced clusters containing just single points or the majority of the data. The other types of linkage are less sensitive to outliers and can split the data into more balanced clusters. The plots in Figure 16 below show the dendrograms from the clustering just performed. They show how unbalanced single linkage is and how it tends to result in long trailing clusters where single observations are fused at each stage.

```
par(mfrow=c(1,3))
plot(hclust.single, main="Single Linkage Dendrogram")
plot(hclust.complete, main="Complete Linkage Dendrogram")
plot(hclust.average, main="Average Linkage Dendrogram")
```

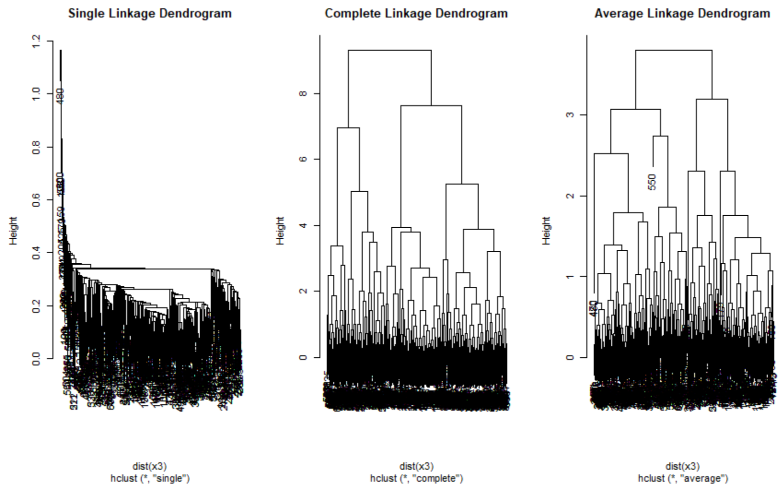


Figure 33: Resulting dendrograms when performing the above code on the noisy dataset.

Before going any further, it's worth verifying what the optimal number of clusters in this dataset is. To do this, the elbow method is again useful, as well as the silhouette coefficient. The function *fviz_nbclust*() from the package *factoextra* is useful in performing these tasks. The method *hcut* is the package's own function for performing hierarchical clustering, combining the clustering and cutting into one function.

```
fviz_nbclust(x3,hcut,method = "wss")
fviz_nbclust(x3, hcut, method="silhouette")
```

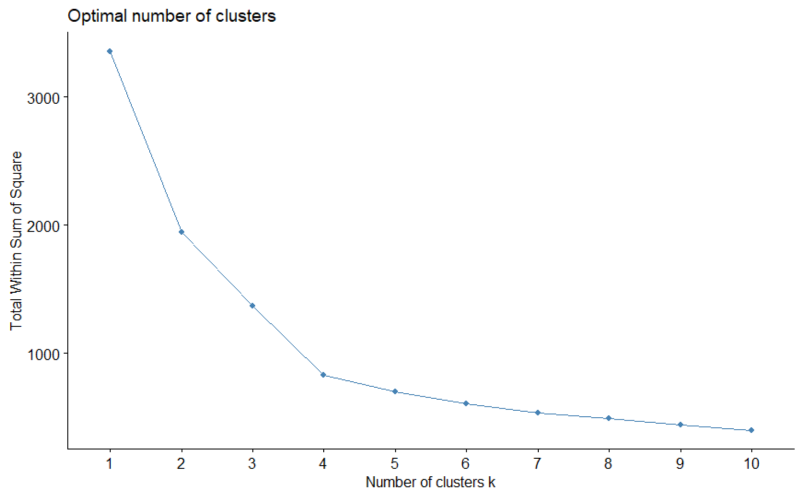


Figure 34: Elbow plot for finding the optimal number of clusters

In Figure 17, the elbow of the curve appears to be at four clusters. This is further verified by the plot of average silhouettes below in Figure 18, where the maximum is at 4. Another method for finding the optimal number of clusters is contained in a useful R package called *NbClust*[26] This function takes a dataset, metric and method, then produces and aggregates a total of 30 indicies used to determine the number of clusters in a dataset. Each index suggests a number for K, then the frequency of each value of K is tallied. The one that is favoured by the most indicies is most likely to be the best number for K.

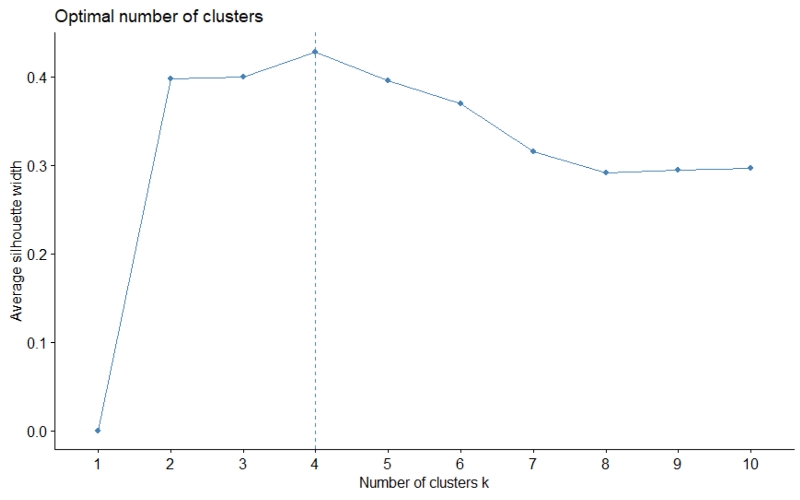


Figure 35: The average silhouette for different numbers of clusterings

```
library(NbClust)
> nb <- NbClust(x3, distance = "euclidean", min.nc = 2,
+              max.nc = 10, method = "average")
#     .....................................
#
#  Among all indices:
#  4 proposed 2 as the best number of clusters
#  2 proposed 3 as the best number of clusters
#  15 proposed 4 as the best number of clusters
#  1 proposed 9 as the best number of clusters
#  1 proposed 10 as the best number of clusters
#
#                        Conclusion
#
#  According to the majority rule, the best number of clusters is 4
```

Therefore out of 30 indicies, the vast majority chose the ideal number of clusters to be 4. However, when the same function is applied to complete linkage, the majority of indicies suggest that the best number of clusters is either 2 or 5.

```
> nb <- NbClust(x3, distance = "euclidean", min.nc = 2,
+              max.nc = 10, method = "complete")
#     .....................................
#
#  Among all indices:
#  7 proposed 2 as the best number of clusters
#  5 proposed 3 as the best number of clusters
#  1 proposed 4 as the best number of clusters
#  7 proposed 9 as the best number of clusters
#  3 proposed 10 as the best number of clusters
#
#                        Conclusion
#
#  According to the majority rule, the best number of clusters is 2
```

The next logical step could be to compare the two types of clustering with their optimal values of K using an evaluation index like the silhouette coefficient or the Dunn Index. The Dunn Index can be computed using the *dunn*() function in the *clvalid*[27] package.

```
# Average Silhouette for Average Linkage
> mean(silhouette(cutree(hclust.average,4),dist(x3))[,"sil_width"])
[1] 0.4688872
# Average Silhouette for Complete Linkage
> mean(silhouette(cutree(hclust.complete,2),dist(x3))[,"sil_width"])
[1] 0.3071899
```

```
library(clvalid)
# Dunn Index for Average Linkage
> dunn(dist(x3),cutree(hc.average, 4))
[1] 0.006880614
# Dunn Index for Complete Linkage
> dunn(dist(x3),cutree(hc.complete, 2))
[1] 0.0003391304
```

Both the internal evaluation measures indicate that average linkage provides a significantly better resultant clustering than complete. In general, the Dunn Index tends to provide higher values for the average linkage so it is often the prefered linkage.

7.3.2 Divisive Clustering

Performing divisive clustering can be done with the *diana*() function in the *Cluster* package [28]. The *diana* function takes a dataset to be clustered, a distance matrix and a metric as arguments. The default metric is euclidian and the distance matrix is inherited from the dataset by default. The function *pltree*() in the *Cluster* package can be used to plot the dendrogram.

```
library(Cluster)
divisive <- diana(x3)
pltree(divisive)
```

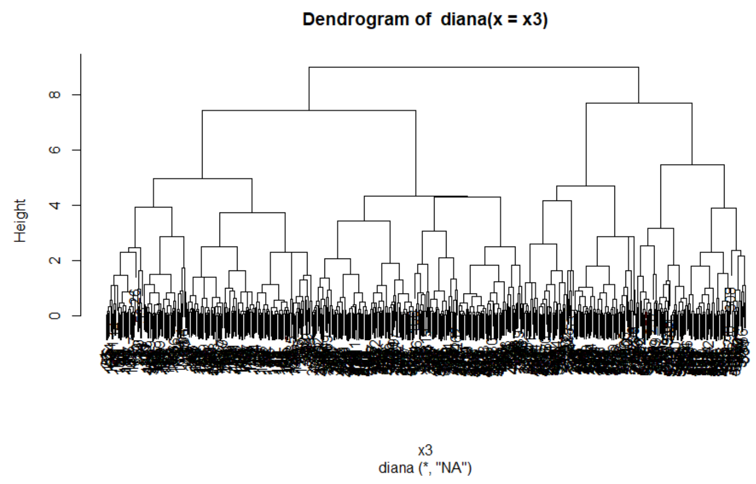


Figure 36: Dendrogram for the divisive clustering on x3

As Figure 19 shows, the clusters seem to be well distributed and evenly matched. Next, the cluster configuration can be shown visually with the *plot*() function.

```
plot(x3, col=cutree(as.hclust(divisive), k=4))
```

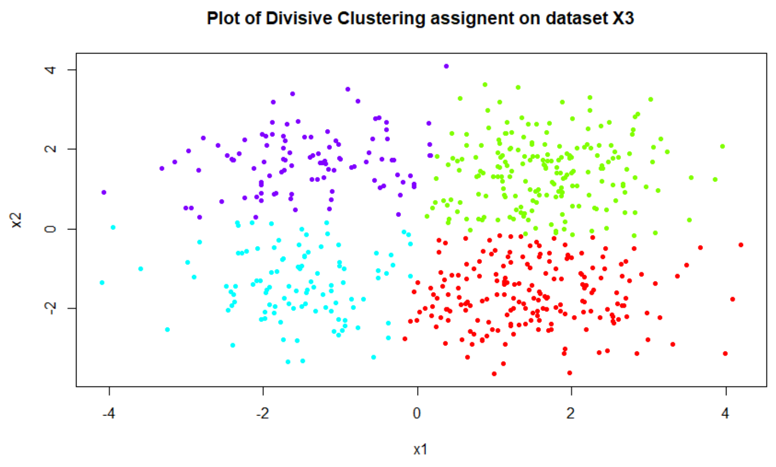


Figure 37: Plot of the divisive clustering on x3

The Dunn index and Average Silhouette can be calculated for this clustering model, in order to get a measure of how well the datapoints fit their assigned clusters.

```
library(clValid)
dunn(dist(x3),cutree(as.hclust(divisive),4))
[1] 0.0168634
mean(silhouette(cutree(as.hclust(divisive),4),dist(x3))[,"sil_width"])
[1] 0.4609537
```

Comparing this back to the agglomerative average-linkage hierarchical clustering model which had a Dunn Index of 0.00688 and Average Silhouette of 0.4688872, the divisive hierarchical model performs very similarly in the average silhouette, meaning on average, the datapoints fit their assigned clusters about the same. However, its Dunn Index is higher, perhaps providing evidence that for this dataset, the divisive clustering model may be better than the agglomerative clustering model.

During the next few weeks, I also intend to compare the 2 main methods by performing them on example datasets in R, and include the R-code and results. Then I will interpret these results to show the strengths and weaknesses of each of these methods.

References

[1] J. MacQueen (1967). *Some methods for classification and analysis of multivariate observations.* Proc. Fifth Berkeley Symp. on Math. Statist. and Prob., Vol. 1 (Univ. of Calif. Press, 1967), 281–297.

[2] Lloyd, Stuart P. (1957). *Least square quantization in PCM*, Bell Telephone Laboratories Paper. Published: Lloyd, Stuart P. (1982)

[3] J.H Ward (1963). *Hierarchical Grouping to Optimize an Objective Function.*

[4] Laurence Morissette and Sylvain Chartier, *Tutorials in Quantitative Methods for Psychology 2013, Vol. 9(1), p. 15-24.*, Université d'Ottawa

[5] N.M Faber, *Generalized rank annihilation method. I: Derivation of eigenvalue problems*, (1994)

[6] C. E. Shannon (1948), *A Mathematical Theory Of Communication.*

[7] T. Calinski and J. Harabasz (1974), *A dendrite method for cluster analysis*

[8] L. Kaufman, P.J. Rousseeuw, et al, *Finding Groups In Data: An Introduction to Cluster Analysis*, 1990

[9] K.Y. Yeung, C. Fraley, A. Murua, A.E. Raftery, and W.L. Ruzzo (2001), *Model-based clustering and data transformations for gene expression data, Bioinformatics.*

[10] Dunn, J.C. (1973) *A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters. Journal of Cybernetics, 3, 32-57*

[11] J. C. Bezdek (1981) *Pattern Recognition with Fuzzy Objective Function Algoritms", Plenum Press, New York*

[12] *Data Clustering: Algorithms and Applications* (2014), edited by C. Aggarwal, C.K. Reddy

[13] *K-Means++: The Advantages of Careful Seeding* (2007), Arthur, D.; Vassilvitskii, S.

[14] Paul S. Bradley, Usama M. Fayyad, *Refining Initial Points for K-Means Clustering* (1998)

[15] Forgy, Edward W. (1965). *Cluster analysis of multivariate data: efficiency versus interpretability of classifications.*

[16] G. W. Milligan, *A Monte Carlo study of thirty internal criterion measures for cluster analysis* (1981)

[17] Hamerly, Greg; Elkan, Charles (2002), *Alternatives to the k-means algorithm that find better clusterings*

[18] Robert L. Thorndike (1953). *Who Belongs in the Family?*. Psychometrika

[19] Hartigan, J.A. (1975), *Clustering Algorithms, New York: John Wiley & Sons, Inc.*

[20] C. Aggarwal, A. Hinneburg, D. A. Keim. (2001), *On the Surprising Behavior of Distance Metrics in High Dimensional Space*

[21] G. James, D. Witten, T. Hastie, R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, (2013)

[22] *SAS/STAT(R) 9.2 User's Guide, Second Edition*, (2011) A. Jones, E. Huddleston

[23] Lampros Mouselimis (2020). ClusterR: Gaussian Mixture Models, K-Means, Mini-Batch-Kmeans, K-Medoids and Affinity Propagation Clustering. R package version 1.2.2. https://CRAN.R-project.org/package=ClusterR

[24] Alboukadel Kassambara and Fabian Mundt (2020). factoextra: Extract and Visualize the Results of Multivariate Data Analyses. R package version 1.0.7. https://CRAN.R-project.org/package=factoextra

[25] M.B. Ferraro, P. Giordani and A. Serafini (2019) fclust: An R Package for Fuzzy Clustering, The R Journal, 11, https://journal.r-project.org/archive/2019/RJ-2019-017/RJ-2019-017.pdf

[26] Malika Charrad, Nadia Ghazzali, Veronique Boiteau, Azam Niknafs (2014). NbClust: An R Package for Determining the Relevant Number of Clusters in a Data Set. Journal of Statistical Software, 61(6), 1-36. URL http://www.jstatsoft.org/v61/i06/.

[27] Guy Brock, Vasyl Pihur, Susmita Datta, Somnath Datta (2008). clValid: An R Package for Cluster Validation. Journal of Statistical Software, 25(4), 1-22. URL https://www.jstatsoft.org/v25/i04/

[28] Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., Hornik, K.(2019). cluster: Cluster Analysis Basics and Extensions. R package version 2.1.0.

[29] Valafar F. Pattern recognition techniques in microarray data analysis. Annals of the New York Academy of Sciences. 2002