Assignment 7

Thijs van der Knaap (s2752077)
Hatim Alsayahani (s3183696)
Sebastian Wehkamp (s2589907)
Dimitris Laskaratos (s3463702)

Group 15

October 22, 2017

1

1.1 Exercise 1.1

1.1. single linkage

Distance Matrix 1

P1	0				
P2	0,9	0			
P3	0,6	0,4	0		
P4	0,5	0,6	0,6	0	
P5	0,7	0,1	0,2	0,3	0
	P1	P2	P3	P4	P5

Distance Matrix 2

P1	0			
P2,5	0,7	0		
P3	0,6	0,2	0	
P4	0,5	0,3	0,6	0
	P1	P2,5	P3	P4

Distance Matrix 3

P1	0		
P2,3,5	0,6	0	
P4	0,5	0,3	0
	P1	P2,3,5	P4

Distance Matrix 4

P1	0	
P2,3,4,5	0,5	0
	P1	P2,3,4,5

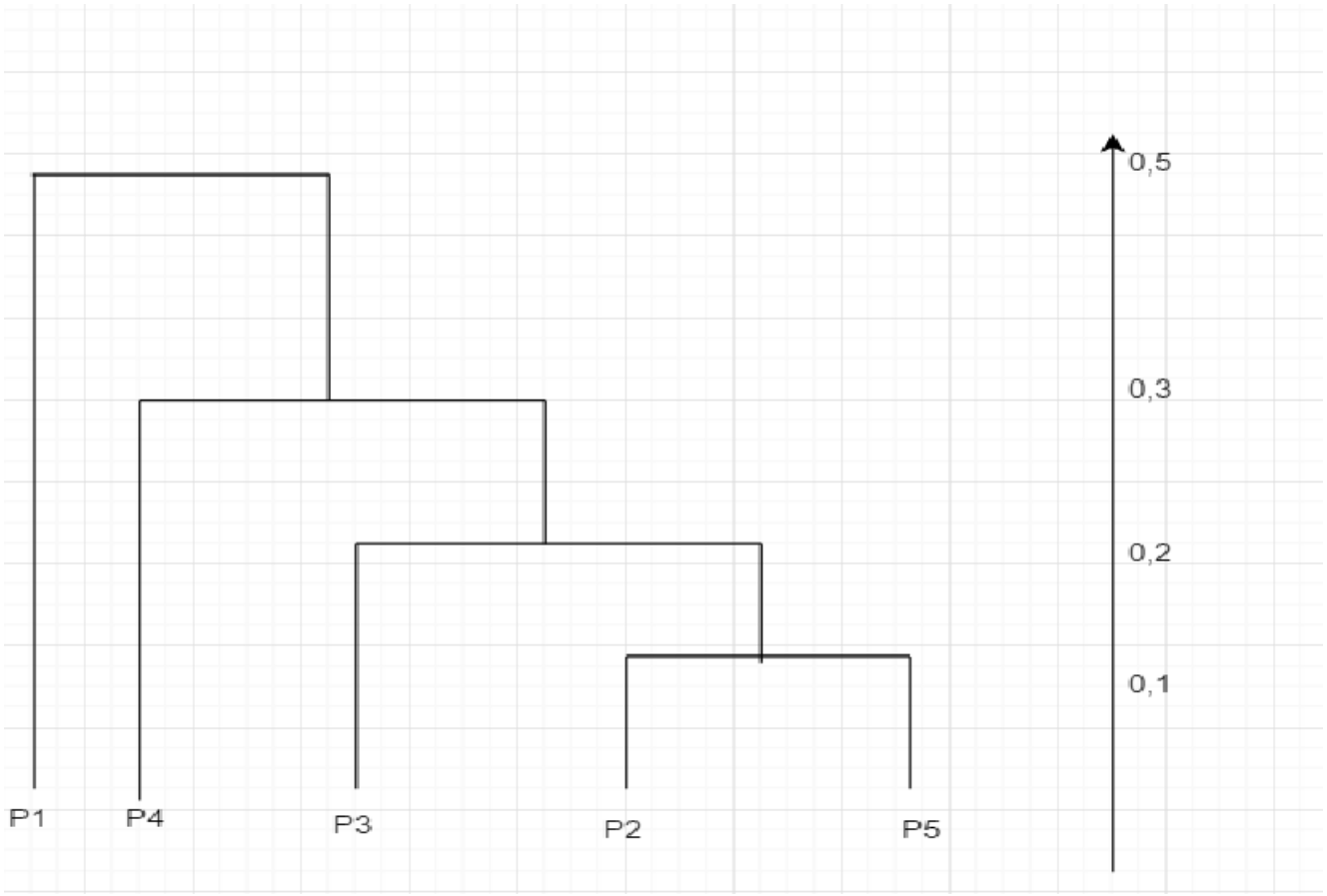


Figure 1: Dendrogram

2 Exercise 1.2

1.2 Average Linkage

Distance Matrix 1

P1	0				
P2	0,9	0			
P3	0,6	0,4	0		
P4	0,5	0,6	0,6	0	
P5	0,7	0,1	0,2	0,3	0
	P1	P2	P3	P4	P5

Distance Matrix 2

P1	0			
P2,5	0,8	0		
P3	0,6	0,3	0	
P4	0,5	0,45	0,6	0
	P1	P2,5	P3	P4

Distance Matrix 3

P1	0		
P2,3,5	0,73	0	
P4	0,5	0,5	0
	P1	P2,3,5	P4

Distance Matrix 4

P1,4	0	
P2,3,5	0,62	0
	P1,4	P2,3,5

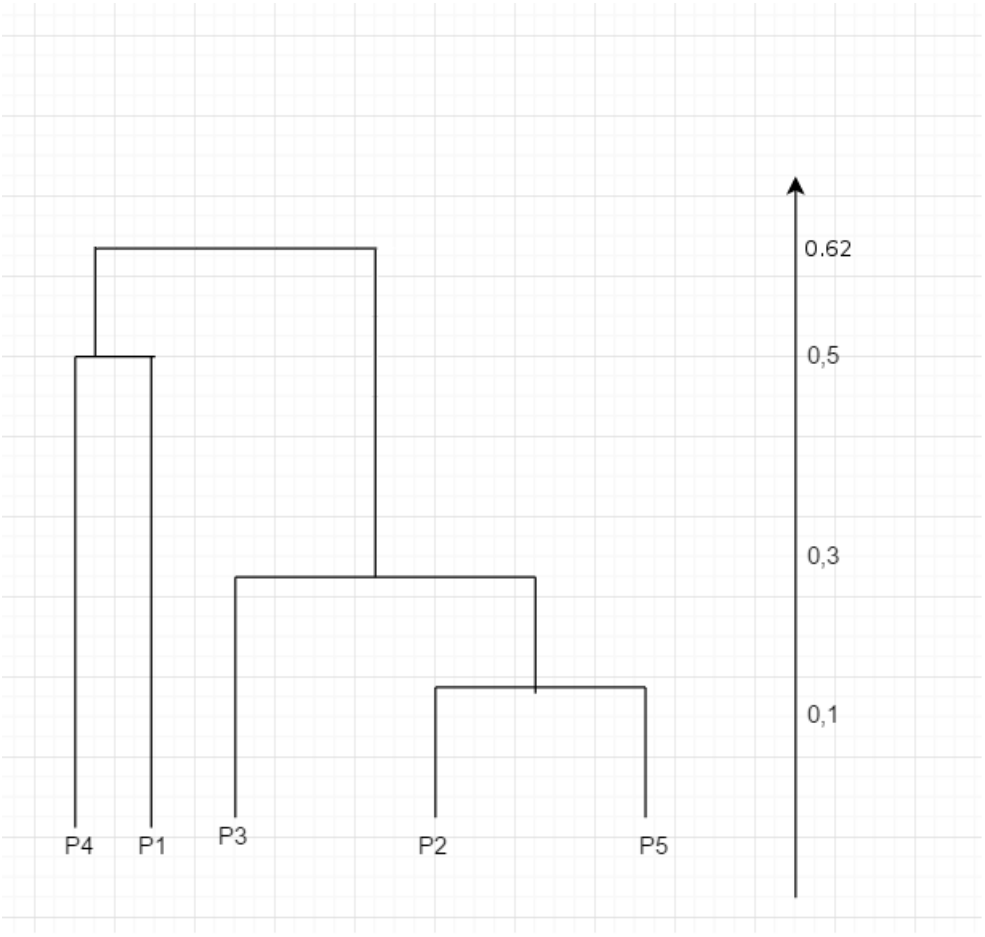


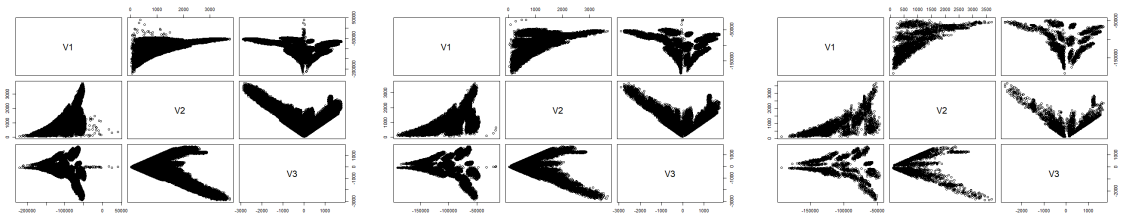
Figure 2: Dendrogram

dist2,5,3=(0.4+0.2)/2*1=0.3 dist4,1,3=(0.6+0.6)/2*1=0.6 dist2,5,4,1=(0.5+0.6+0.3+0.7)/2*2=0.525 Because deist2,5,3 is smaller than dist4,1,3 and dist2,5,4,1, clusters2,5,3 are merged at the second stage.

2

2.1

The first thing we did was plotting the 3-dimensional dataset. This plot can be seen in Figure 3a. When plotting the data we immediately noticed that the size of the dataset might become a problem. There are 280 000 entries in the dataset and just creating a simple plot of this took a while. We concluded that the size of the dataset was too large and doing an analysis directly on the dataset is not feasible for more complex algorithms.



(a) Plot of the original 3-dimensional dataset

(b) Plot of the sampled (28000) 3-dimensional dataset

(c) Plot of the sampled (2800) 3-dimensional dataset

Figure 3: Comparison of the original dataset versus the sampled datasets

So we have to reduce the size of the dataset in some way. To do this we created a sample of $\frac{1}{10}$th and a sample of $\frac{1}{100}$th of the original size using random sampling, the implementation can be found on on page 15. This resulted in the plots from Figure 3b and Figure 3c. You can clearly see that the structure is the same for all plots even when we reduced the size by a factor of 100. You do however lose information from inside the clusters however the general outlines are the same.

2.2

For this exercise we had to figure out what the difference was between the 3-dimensional and 6-dimensional dataset. The first thing we did was to generate the Hopkins statics as the assignment stated. We did this with the code located on page 15 and the results can be seen in Table 1. The results from the Data3 dataset are very close to zero meaning that the dataset is quite uniformly distributed. The data6 results are still a bit low but are on average about 0.2 meaning that it is a bit less uniformly distributed.

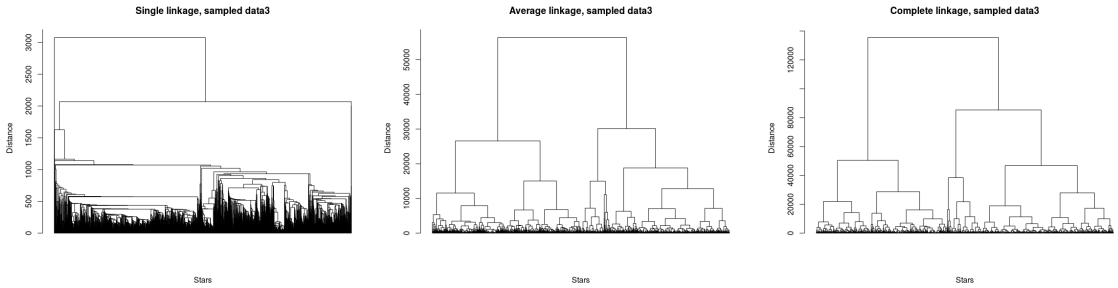
Table 1: Results of the Hopkins statics on 2800 sample

Dataset\Excution NR	1	2	3	4	5
Data3	0.07275288	0.08226438	0.1775606	0.1233985	0.09082806
Data6	0.1956465	0.1892778	0.2219941	0.1817965	0.1901705

2.3/2.4

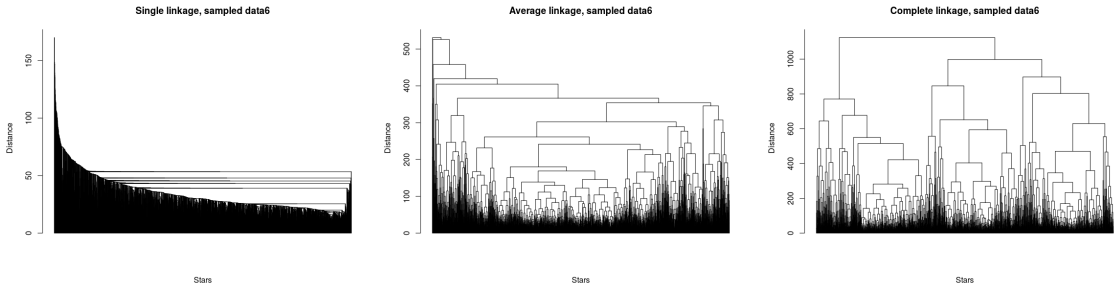
In this section we created some dendograms for both datasets using our sample of size 2800. We experimented with the linkage for which we plotted single linkage, average linkage and complete linkage. The dendograms for the 3-dimensional data can be seen in Figure 4 and the dendograms for the 6-dimensional dataset are in Figure 5. The code to generate the dendograms can be found on page 15.

Using single linkage Figure 5a and in Figure 4a we can see that the 6-dimensional dataset is more like a single link meaning that its more like a single cluster instead of smaller distinct clusters compared to the 3-dimensional dataset. Using average linking the results for the 3-dimensional dataset look good and are the partitions are nicely split in half each time. This is different from the 6-dimensional dataset since in Figure 5b you clearly see there is more data on the left compared to the right. This problem is solved when we use Complete linkage where both datasets produce nice results as can be seen in Figure 4c and Figure 5b. In the 6-dimensional dataset the datacluster on the left has moved nicely to the center producing a better dendogram.



(a) Single linkage (b) Average linkage (c) Complete linkage

Figure 4: Dendograms for Data3



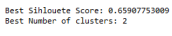
(a) Single linkage (b) Average linkage (c) Complete linkage

Figure 5: Dendograms for Data6

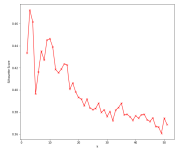
3

3.1

We will compare the two datasets using the Silhouette coefficient after clustering with K-Means. After normalising the 2800 sample data, we apply KMeans and then compute the maximum Silhouette score for the datasets. For *data3.txt* we get Figure 6 and for *data6.txt* we get Figure 7.

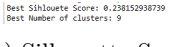


(a) Silhouette Score

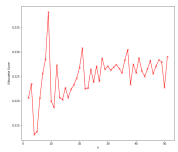


(b) Silhouette scores for all k

Figure 6: Data3



(a) Silhouette Score



(b) Silhouette scores for all k

Figure 7: Data6

We now have a perspective into the density and the separation of the formed clusters. *data3* has been separated into two clusters with an average Silhouette score of 0.66. *data3* has been separated into nine clusters with an average score of 0.24. This means two things:

1. When we cluster the data by taking into account the three features from data3 (energy, angular momentum, angular momentum along the z-axis), the algorithm's decision is much clearer, as the score suggests. The elements are closer to their assigned clusters than far from it. On the contrary, the score for data6 suggests that many elements stand very close to the decision boundary between neighbouring clusters.

2. The fact that the sample of data3 are divided in two clusters means that the correlation between them is quite high, while the same cannot be said for the sample of data6 which is divided in 9 clusters. The features of data3 might suggest that two stars can be in the same cluster when judging from their energy, for example, even if their positions may differ greatly.

3.2

In order to find the best number of clusters (k) in *data3 and data6*, first we apply the KMeans clustering algorithm, from the *sklearn* library. Subsequently, several evaluation measurements are applied on the cluster set to determine k.

In the code in page 16, we apply the Silhouette coefficient on the 2800 randomly selected points from the *data6* set. The coefficient is calculated for every cluster number from 2 to 50. This gives the result shown in Figure 9

```
Best Sihlouete Score: 0.238688713158
Best Number of clusters: 10
```

Figure 8: Silhouette Best Score

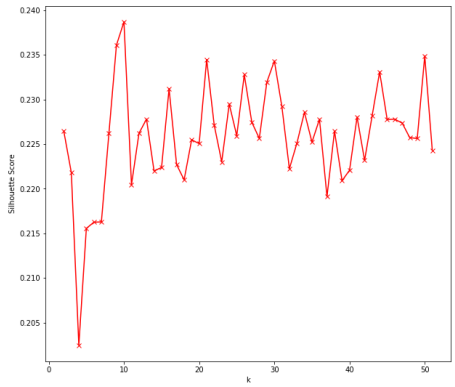


Figure 9: Silhouette measurements

It's notable that the Silhouette method gets the same results for larger (eg. 4000) or smaller samples and even in different cluster ranges (eg. 2 to 20), which makes it quite reliable.

Note: By running the script several times, the score itself remains unchanged , for data6, but the k might increase or decrease. What's important is that the majority of iterations generate a k=9 or k=10, and so we keep that. What's not expected, though, is that k is always 2 for data3.

Additionally, we use the following metrics to evaluate the KMeans clustering; **Centroid Distances**. In this method we will calculate the sum of distances of all the elements from the central element of each cluster found by KMeans. In the end, for each cluster number we will also have an accumulated cost (sum of distances). Code is on page 16. The results from running the code are:

```
k: 2   cost: 1042.43633858
k: 3   cost: 930.644676415
k: 4   cost: 851.422340193
k: 5   cost: 787.297806711
k: 6   cost: 733.718886873
k: 7   cost: 688.394188741
k: 8   cost: 643.640575323
k: 9   cost: 610.434114615
k: 10   cost: 581.053083521
k: 11   cost: 558.853547518
k: 12   cost: 539.324749005
k: 13   cost: 521.708034259
k: 14   cost: 505.838340913
k: 15   cost: 492.862976017
k: 16   cost: 480.207086059
k: 17   cost: 469.169989151
k: 18   cost: 458.543950676
k: 19   cost: 449.707655282
k: 20   cost: 441.790642314
k: 21   cost: 431.392084119
```

(a)

```
k: 20   cost: 441.790642314
k: 21   cost: 431.392084119
k: 22   cost: 424.179993822
k: 23   cost: 415.94869644
k: 24   cost: 408.774811614
k: 25   cost: 402.979351327
k: 26   cost: 396.063323852
k: 27   cost: 389.33130141
k: 28   cost: 383.449948424
k: 29   cost: 378.454676087
k: 30   cost: 373.784748871
k: 31   cost: 368.666458886
k: 32   cost: 362.867832165
k: 33   cost: 357.626370674
k: 34   cost: 352.124828715
k: 35   cost: 348.984550953
k: 36   cost: 344.726370664
k: 37   cost: 339.982804421
k: 38   cost: 336.730874727
k: 39   cost: 331.966499664
k: 40   cost: 328.530943245
k: 41   cost: 323.992939938
```

(b)

```
k: 40   cost: 328.530943245
k: 41   cost: 323.992939938
k: 42   cost: 321.20706093
k: 43   cost: 316.071115031
k: 44   cost: 313.961896728
k: 45   cost: 312.322601116
k: 46   cost: 309.222858375
k: 47   cost: 307.164473766
k: 48   cost: 304.114197268
k: 49   cost: 299.975000659
k: 50   cost: 297.97027664
k: 51   cost: 296.032736894
```

(c)

Figure 10: Cost for each k with centroid distances.

(a) (b)

Figure 12: data3 and data6

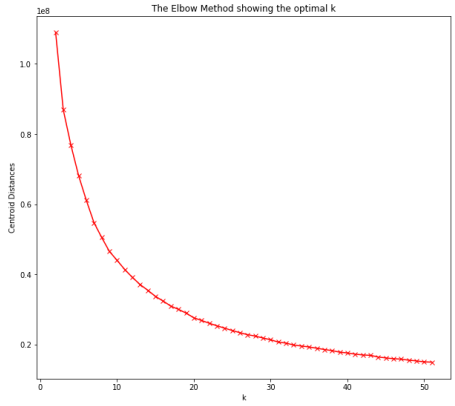


Figure 11: Silhouette Best Score

As we can observe from figures 10 and 11, for data6 the costs decreases rapidly around cluster sizes 14 and 16, and after that the change is much slower. For data3, the equivalent k is 10 as seen in Figure 12

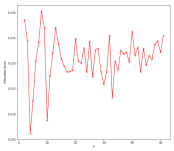
Sum of Square Errors (SSE)

In the code in page 17 we use the Sum of Square Errors to evaluate the classification of KMeans. What we need is a k at which the the Sum has a slower decrease rate, as k increases. We get the following plots for data3 and data6 respectively in figure 13:

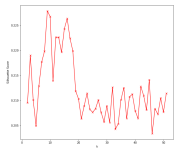
For data3 we can see that the error rate starts to decrease slower at k=10, while for data3 it is 4.

(a) data3 (b) data6

Figure 13: Sum of Square Errors for each k.

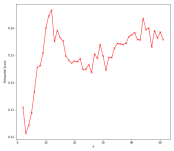


(a) Euclidean

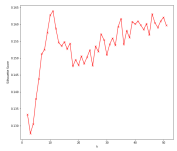


(b) Manhattan

Figure 14: Silhouette Score when using Euclidean and Manhattan



(a) random



(b) kmeans++

Figure 15: Random initialization vs. kmeans++ initialization

From all of the above, we can select k=10 for data6, but it is unclear what it should be for data3 as the previous methods have failed to converge to a certain point.

3.3

Using a different distance method on the Silhouete score like Manhattan, the maximum score doesn't change much but the set of scores from all the iterations changes quite a bit. Typically, in Euclidean the average score is higher than the average score using Manhattan. A comparison can be seen in figure 14.

3.4

So far, the initialization of the cluster has been totally random, selecting 2800 from the entire dataset and then performing KMeans on it. In this case, we are gonna use a different initialization to calculate the Silhouette score on data6. In the command $kmeans = KMeans(n_clusters = i, random_state = 0)$ in the code in page 16 we use the parameter $init =' random'$ and $init =' k-means++'$ as different initializations. The results are respectively (a) and (b) in figure 15.

As we can observe, there is no apparent change in best cluster number returned or the max Silhouette score.

4

4.1

To use this pseudo code, two equations are needed listed in Equation 1 and Equation 2 where

$$\begin{aligned}
\gamma_{ik}^{(m)} &= \textit{the responsibility of Gaussian } k \textit{ for sample} x_i \\
w_k^{(m)} &= \textit{The probability density function of a multivariate Gaussian for cluster } k \\
w_l^{(m)} &= \textit{Gaussian for cluster } l \\
N(x_i|\mu_l, \sum_l) &= \textit{The fraction of the dataset belonging to cluster } l
\end{aligned}$$

$$\mu_k^{new} = New\ weight\ for\ cluster$$
$$N_k = all\ points\ in\ cluster$$
$$\sum_{i=l}^{N} = Sum\ over\ all\ points$$
$$w_{ik} = Weight\ of\ point\ in\ cluster$$
$$x_i = General\ weight\ of\ point$$

$$\gamma_{ik}^{(m)} = \frac{w_k^{(m)} N(x_i|\mu_k, \sum_k)}{\sum_{l=1}^{k} w_l^{(m)} N(x_i|\mu_l, \sum_l)} \tag{1}$$

$$\mu_k^{new} = (\frac{1}{N_k} \sum_{i=l}^{N} w_{ik} * x_i \tag{2}$$

```
Initialisation:

Select random datapoints as initial means
Covariance cluster = Covariance training set
Equal (prior) probability

EM:
loop until convergence:
  #Expectation
  for every cluster:
    for every datapoint:
      Matrix x = Calculate Probability Density Function (pdf) using eq1
  Determine weighted probabilities each cluster using Matrix x and prior
      probabilities

  #Maximisation
  Store previous means in order to check for convergence
  for every cluster
    calculate weighted average using eq2

  In case previous means equal current means convergence = true
end
```

To analyse the star data we choose to combine data3 and data6, which resulted in an 9 dimensional data set.

4.2

The mclust implementation of R automatically uses fourteen different distributions for ($\sum_k$). These can be found on here. μ_k is automatically initialised by mclust by using agglomerative clustering as the initial setup. So we do not have to set μ_k and we can choose the best working $\sum_k$ afterwards.

Looking at the BIC plot in figure 16 we see that the highest BIC value acquired is -300.000

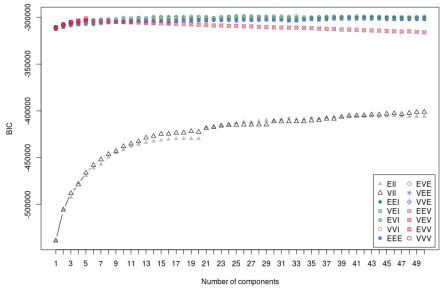
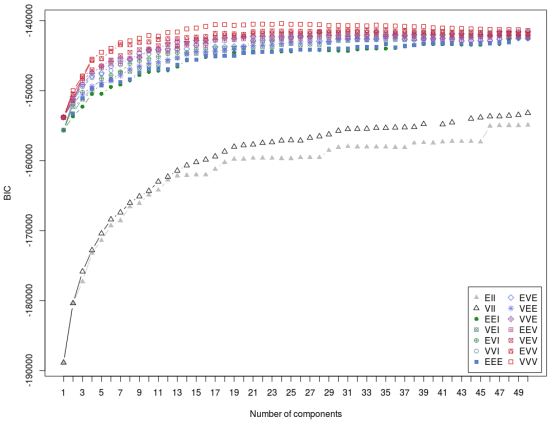
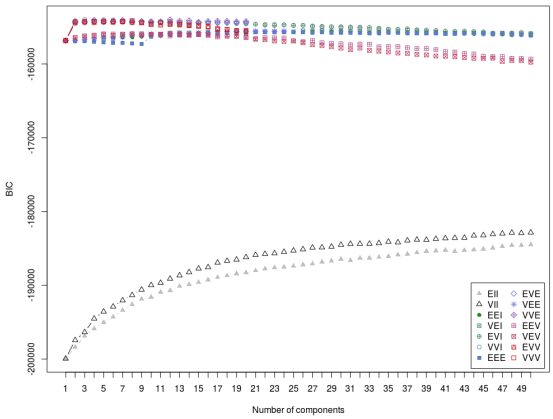


Figure 16: BIC data3 and data6 merged

If we perform the BIC analysis on figure 17a and figure 17b which are the separate analysis of data3 and data6. for data3 this gives us a best BIC value of -142.000 and for data6 a BIC value of -155.000.



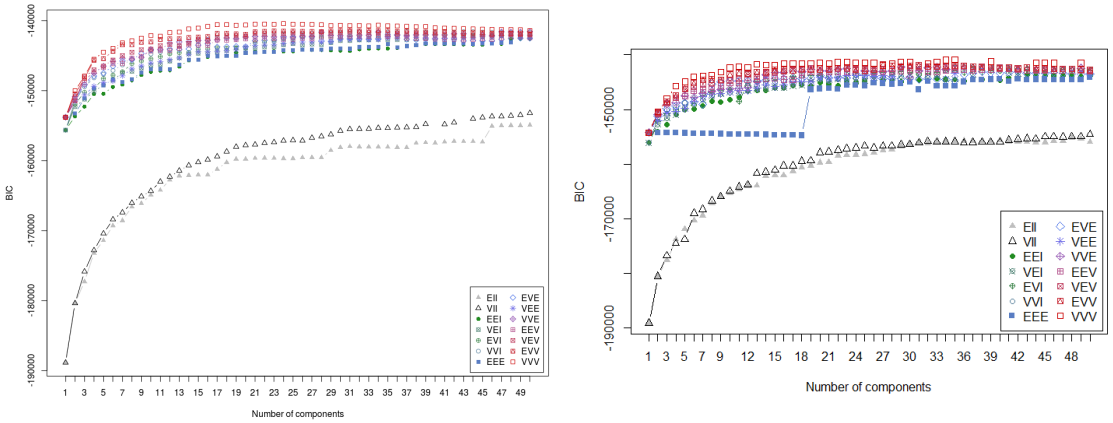
(a) BIC data3



(b) BIC data6

data3 and data6 are outperforming the merged data-set. data3 is doing a little better than data6. therefore we will use the VVV clustering from data3 to which has the highest value with 17 clusters. VVV clustering means the the clusters are ellipsoidal and can vary in volume, shape and orientation.

As stated before, mclust uses agglomerative clustering as a default initialisation. As an experiment we tried to initialise the data with random points as initialisation. A comparison between both agglomerative and random initialisation can be seen in Figure 18. Although the difference is not that large, the initialisation as can be seen in Figure 18a performs better than the random initialisation from Figure 18b since the BIC value of agglomerative initialisation is higher.



(a) BIC data3 with agglomerative initialisation

(b) BIC data3 with random initialisation

Figure 18: Comparison between different initialisation methods

4.3

In this exercise we will determine "Which cluster of stars could arguably be part of a galaxy which most recently merged and which stars might be part of a galaxy of a very early merge." According to the exercise as well, galaxies that recently merged are less diffuse than galaxies that have been merged for a longer amount of time. Therefore we will look through the clusters and see which clusters have a low cohesion. However first we will apply a dimensionality reduction technique on the GMM model and plot it as a scatter plot. Hopefully we can use this to spot the clusters difference in cohesion easily.

In Figure 19 you can see the scatterplot we created using the dimensionality reduction technique. The code to generate this plot is listed on page 18. The cluster we want to look for has a very low cohesion so is spread out the most. A possible candidate for the new cluster would be the green asterisks on the scatterplot since they have a very large spread. When we look at Figure 20 we can clearly see that that cluster corresponds to cluster 8 which has the lowest density by far. From these images you clearly see that since it has the largest spread it also has the lowest cohesion.

So now we want to know which stars from our sample belong to cluster 8. Using the code located on page 18 a CSV is generated which contains all of our qualifications of our 2800 sample. In the CSV file you can see which stars belong to cluster 8 and thus are part of the oldest galaxy.

Looking at the same figure we will list all cluster from old to new:

1. *8*
2. *11*
3. *10*
4. *9*
5. *16*
6. *3*
7. *13*
8. *17*
9. *1*
10. *14*
11. *6*
12. *5*

13. *2*
14. *4*
15. *7*
16. *12*
17. *15*

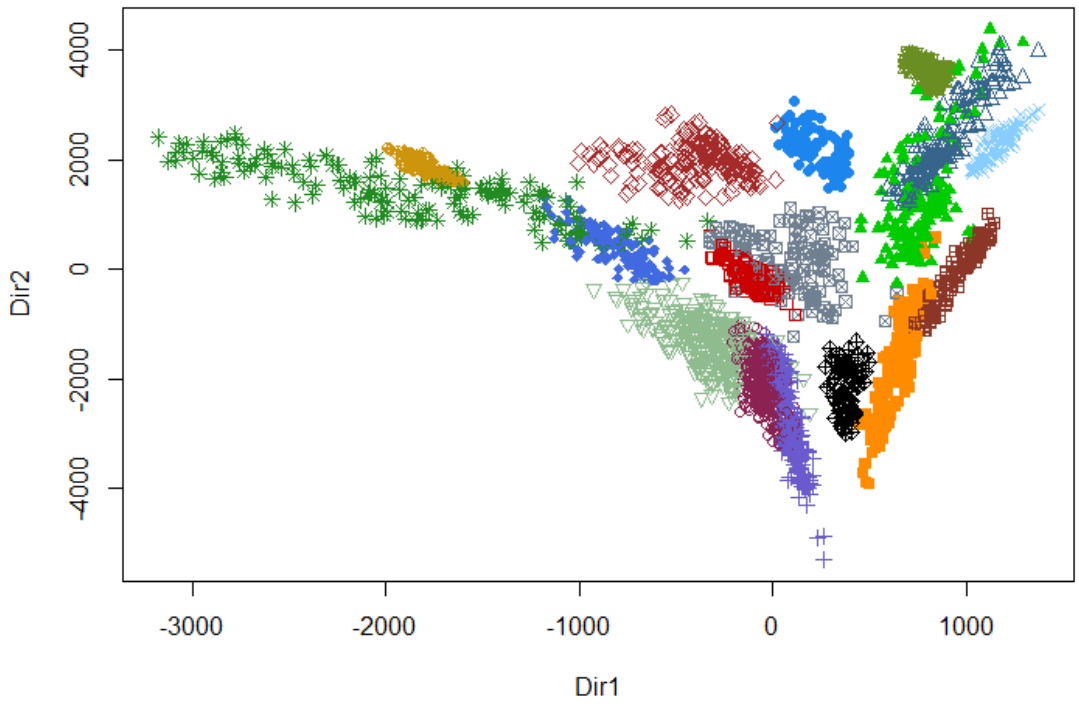


Figure 19: Scatter plot created using GMM and dimensionality reduction

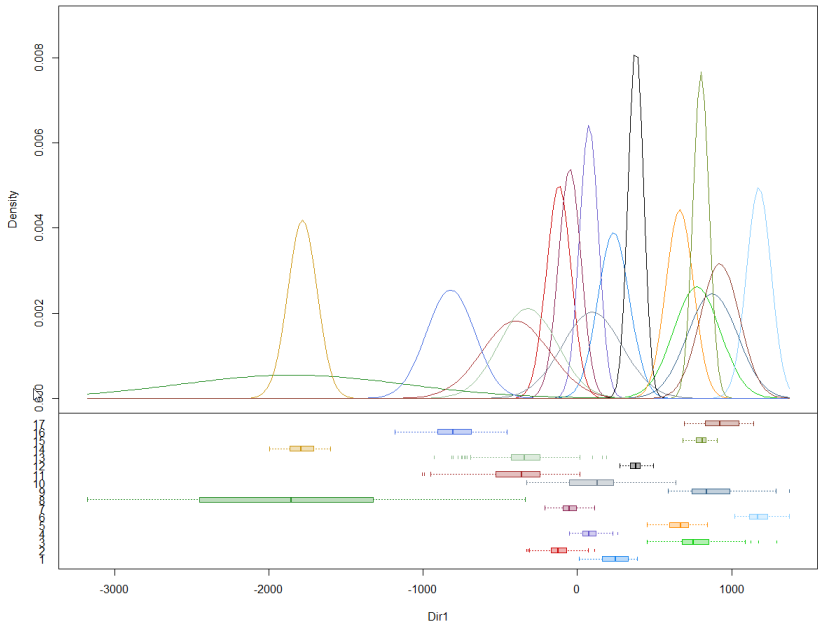


Figure 20: Density plot created using GMM and dimensionality reduction

Appendix

2.1

```
data3 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

sample <- data3[sample(nrow(data3), 28000, replace=FALSE),]

plot(sample)
```

Listing 1: Code to create a sample of the original dataset

2.2

```
library("clustertend")
data3 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)
data6 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

sample <- data3[sample(nrow(data6), 2800, replace=FALSE),]

hop <- hopkins(sample, 10, header=T)
```

Listing 2: Code to generate the Hopkins statics

2.3

```
#Set working directory to directory of the datasets
setwd("/media/HDD-Thijs/Schooldocumenten/2017-2018/IntroDataScience/
    assignment7")

#read Data from csv
data3 <- read.csv("data3.csv")
data6 <- read.csv("data6.csv")

#sample Data
sampledData3 <- data3[sample(nrow(data3), 2800), ]
sampledData6 <- data6[sample(nrow(data6), 2800), ]

dist3 <- dist(sampledData3, method = "euclidean") # distance matrix
dist6 <- dist(sampledData6, method = "euclidean") # distance matrix

#create plots
singleData3 <- hclust(dist3, method="single")
plot(singleData3, labels=FALSE, hang = -1, main="Single linkage, sampled
    data3", xlab="Stars", ylab="Distance", sub="")

singleData6 <- hclust(dist6, method="single")
plot(singleData6, labels=FALSE, hang = -1, main="Single linkage, sampled
    data6", xlab="Stars", ylab="Distance", sub="")

singleData3 <- hclust(dist3, method="complete")
plot(singleData3, labels=FALSE, hang = -1, main="Complete linkage, sampled
    data3", xlab="Stars", ylab="Distance", sub="")

singleData6 <- hclust(dist6, method="complete")
plot(singleData6, labels=FALSE, hang = -1, main="Complete linkage, sampled
    data6", xlab="Stars", ylab="Distance", sub="")

```

```
singleData3 <- hclust(dist3, method="average")
plot(singleData3, labels=FALSE, hang = -1, main="Average linkage, sampled
    data3", xlab="Stars", ylab="Distance", sub="")

singleData6 <- hclust(dist6, method="average")
plot(singleData6, labels=FALSE, hang = -1, main="Average linkage, sampled
    data6", xlab="Stars", ylab="Distance", sub="")
```

Listing 3: Code to create dendograms

3.2

```
import csv
import numpy as np
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import random
from matplotlib import pyplot as plt

reader = csv.reader(open("data6.csv", "r"), delimiter=",")
next(reader, None)
lines = [line for line in reader]
randomLines = random.sample(lines, 4000)
x = list(randomLines)
result = np.array(x).astype("float")

sampleMetrics=[]

for i in range(2, 52):
    kmeans = KMeans(n_clusters=i)
    cluster_labels=kmeans.fit_predict(result)
    sampleMetrics.append((silhouette_score(result, cluster_labels, metric='
        euclidean')))

bestScore=max(sampleMetrics)
bestCluster=sampleMetrics.index(bestScore)
print("Best Sihlouete Score: "+str(bestScore))
print("Best Number of clusters: "+str(bestCluster+2))

fignum=1
fig = plt.figure(fignum, figsize=(10, 9))
plt.plot(range(2, 52), sampleMetrics, 'rx-')
plt.xlabel('k')
plt.ylabel('Silhouette Score')

fig.show()
```

Listing 4: Python code to find the best number of clusters using Silhouette Score. k.

```
import csv
import numpy as np
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
import random
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler(feature_range=(-1, 1))
reader = csv.reader(open("data6.csv", "r"), delimiter=",")
next(reader, None)
lines = [line for line in reader]
randomLines = random.sample(lines, 2800)
x = list(randomLines)
```

```
result = np.array(x).astype("float")
scaler.fit_transform(result)

sampleMetrics=[]
distances = []

for i in range(2, 52):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit_predict(result)
    distances.append(kmeans.inertia_)
    print("k: " +str(i)+"   cost: "+str(kmeans.inertia_))


fignum=1
fig = plt.figure(fignum, figsize=(10, 9))
plt.plot(range(2, 52), distances, 'rx-')
plt.xlabel('k')
plt.ylabel('Centroid Distances')
plt.title('The Elbow Method showing the optimal k')

fig.show()
```

Listing 5: Python code to find the best number of clusters with centroid distances.k.

```
library(cluster)
library(HSAUR)

set.seed(381)

data3 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)
data6 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Prepare data for merge and merge into dataTotal
data3$id <- c(1:nrow(data3))
data6$id <- c(1:nrow(data6))
dataTotal <- merge(data6, data3, by = "id")
drops <- c("id")
data3 <- data3[ , !(names(data3) %in% drops)]
data6 <- data6[ , !(names(data6) %in% drops)]
dataTotal <- dataTotal[ , !(names(dataTotal) %in% drops)]

#sample Data
sampleSelector <- sample(nrow(data3),2800)
sampledData3 <- data3[sampleSelector, ]
sampledData6 <- data6[sampleSelector, ]
sampledDataTotal <- dataTotal[sampleSelector, ]

mydata <- sampledData3
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
for (i in 2:52) wss[i] <- sum(kmeans(mydata,
                                     centers=i)$withinss)
plot(1:52, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")

mydata <- sampledData6
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
for (i in 2:52) wss[i] <- sum(kmeans(mydata,
                                     centers=i)$withinss)
plot(1:52, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")
```

Listing 6: Code to calculate optimal number of clusters using SSE

4.2

```
require(mclust)

set.seed(381)

data3 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)
data6 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Prepare data for merge and merge into dataTotal
data3$id <- c(1:nrow(data3))
data6$id <- c(1:nrow(data6))
dataTotal <- merge(data6, data3, by = "id")
drops <- c("id")
data3 <- data3[ , !(names(data3) %in% drops)]
data6 <- data6[ , !(names(data6) %in% drops)]
dataTotal <- dataTotal[ , !(names(dataTotal) %in% drops)]

#sample Data
sampleSelector <- sample(nrow(data3),2800)
sampledData3 <- data3[sampleSelector, ]
sampledData6 <- data6[sampleSelector, ]
sampledDataTotal <- dataTotal[sampleSelector, ]

#Create BIC plot and statistics
BIC = mclustBIC(sampledData3, G=1:50)
plot(BIC)
summary(BIC)

#Create model using previously created BIC and VVV as model
modDefault = Mclust(sampledData3, x=BIC, modelName = "VVV")
summary(modDefault, parameters = TRUE)
plot(modDefault, what = "classification")

#Create model using random initialization
randPairs <- randomPairs(sampledData3)
str(randPairs)
modRand <- Mclust(sampledData3, initialization = list(hcPairs = randPairs),
     G=1:50)
plot(modDefault, what="BIC")
```

Listing 7: R code to generate the likelihood function data from both random initialisation and from agglomerative initialisation

4.3

```
require(mclust)

set.seed(381)

data3 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)
data6 <- read.csv(file.choose(new = FALSE), header=TRUE, stringsAsFactors=
    TRUE)

#Prepare data for merge and merge into dataTotal
data3$id <- c(1:nrow(data3))
data6$id <- c(1:nrow(data6))
dataTotal <- merge(data6, data3, by = "id")
drops <- c("id")
```

```r
data3 <- data3[ , !(names(data3) %in% drops)]
data6 <- data6[ , !(names(data6) %in% drops)]
dataTotal <- dataTotal[ , !(names(dataTotal) %in% drops)]

#sample Data
sampleSelector <- sample(nrow(data3),2800)
sampledData3 <- data3[sampleSelector, ]
sampledData6 <- data6[sampleSelector, ]
sampledDataTotal <- dataTotal[sampleSelector, ]

#Create model using previously created BIC and VVV as model
BIC = mclustBIC(sampledData3, G=1:17)
modDefault = Mclust(sampledData3, x=BIC, modelName = "VVV")
plot(modDefault, what = "classification")

#Apply dimensionility reduction and create scatter plot
modDR = MclustDR(modDefault)
plot(modDR)


# add a column in myData CLUST with the cluster.
sampledData3$CLUST <- modDefault$classification

# now to write it out:
write.csv(sampledData3[,c("CLUST","V1","V2","V3")], # reorder columns to
    put CLUST first
          file="out.csv",                    # output filename
          row.names=FALSE,                   # don't save the row numbers
          quote=FALSE)
```

Listing 8: R code applying GMM and dimensionality reduction in order to create some plots and export the classification to a CSV