BDBA: Weekly Assignment 3

Rutger Smid, 487148 & LoÃ¯c Roldan Waals 409763

9-4-2018

Question 1

(a) Compute and report the centroid for each initial cluster.

We first compute the X1 and X2 values for the two centroids as follows:

Computing Centroid 1:

X1 = (1+5+6)/3 = 4

X2 = (4+1+2)/3 = 2,33

Computing Centroid 2:

X1 = (1+0+4)/3 = 1,66

X2 = (3+4+0)/3 = 2,33

(b) Reassigning observations

First the distance of each observation to the two centroids is calculated. The distance from *Centroid 1* to *observation 1* is calculated: sqrt((4 - 1)ˆ2 + (2.33 - 4)ˆ2) = 3.43. The same method is used to calculate the distance to *Centroid 2*, which gives us a value of 1.80. This allows us to conclude that *observation 1* is closest to *Centroid 2* and thus it will be assigned to it.

The same method is used for all the other observations, the new cluster labels are:

- observation 1 = Centroid 2
- observation 2 = Centroid 2
- observation 3 = Centroid 2
- observation 4 = Centroid 1
- observation 5 = Centroid 1
- observation 6 = Centroid 1

(c) Recalculating cluster centroids

Computing Centroid 1: X1 = (5+6+4)/3 = 5 X2 = (1+2+0)/3 = 1

Computing Centroid 2: X1 = (1+1+0)/3 = 0,66 X2 = (4+3+4)/3 = 3,66

When reassigning the observations to the centroids we find that none changed. This means that the final values for *Centroid 1* are X1 = 5, X2 = 1 and that the final values for *Centroid 2* are X1 = 0.66, X2 = 3.66. The cluster labels are the same as in **Question 1 (a)**:

- observation 1 = Centroid 2
- observation 2 = Centroid 2
- observation 3 = Centroid 2
- observation 4 = Centroid 1
- observation 5 = Centroid 1
- observation 6 = Centroid 1

Question 2

Accuracy is a simple performance measuren and not suitable for classificating imbalanced data sets Accuracy simply divides the number of correct predictions by the total nbumber of predicitons. Although it can give an indicaiton of the quality of the model, it should always be compared with the baseline accuracy of a model/dataset. Let's say you want to build a model which diagnoses whether a patient has disease X. It might not be a common disease so 999 out of 1000 patients will not have this disease, this is an example of an imbalanced data set. Then, a model which simply predicts that no patients have this disease will yield an accuracy of 0.999%. This sounds impressive, but the model is not very useful in practice. Looking back at Week 1's assignment, the final model had a accuracy of 70% by predicting US for all observations. This was simply due to the fact that 70% of the population would travel to the US.

An implication of this problem is that for imbalanced data sets, another performance metric should be used. A solution to this problem is the Receiver Operating Characteristic (ROC) curve which allows the comparison of models withouth knowing the class distribution. The ROC curve is plotted using thee true positive rate (TPR) against the false positive rate (FPR). The ROC curve allows us to compare different models and interpret their performance easily, even in the case of imbalanced data sets.

Question 3

Twitter You can also embed plots, for example:

Question 4

As a first step, we install all necessary packages in R. Subsequently we import the dataset and remove the first colomn. This colomn is simply an index variable which is redundant as each row already has an index in R.

```
library(cluster)
library(class)
library(C50)
library(rpart)
library(rpart.plot)
```

```
#Loading Data & removing fist row (index variable)
library(readr)
spotify <- read_csv("spotify.csv", col_types = cols(X1 = col_skip()))
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
spotify <- spotify[complete.cases(spotify),]
```

After loading the data, a first look at the dataset shows that there are some missing variables. These 68 instances are dleted from the dataset.

```
# Inspecting Data
str(spotify)
summary(spotify)
colSums(is.na(spotify))
```

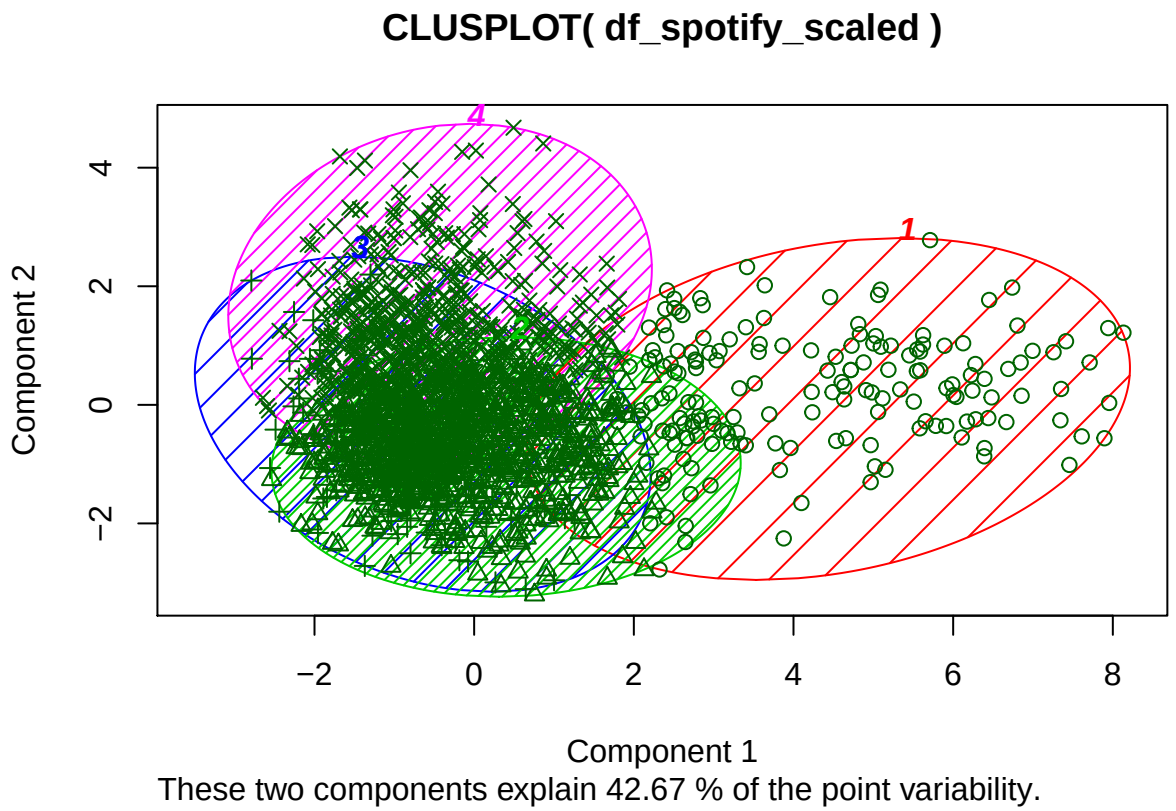
```
# Removing 68 NA instances
spotify <- spotify[complete.cases(spotify) ,]
```

Since we are going to perform K means clustering, we are only going to use the numerical variables in this dataset. To prevent skewed/imbalanced data to impact the model, these numerical variables have to be scaled. The variables *key*, *mode*, *timesignature* and *target* are integer variables but will not be used in this model as these are categorical variables according to Spotify (2018).

```
# Scaling all numerical variables
spotify_num <- spotify[,c(1:5 ,7,8,10,11,13)]
spotify_scaled <- scale(spotify_num)
df_spotify_scaled <- as.data.frame(spotify_scaled) #i believe this step is unnecessary
```

With K-means clustering, a number of clusters K has to be set arbitrarily. We are going to iteratively select K clusters and inspect the results using Principal Component Analysis. This method plots observations together on their scores on the two "most important" variabels: the principal components. These are defined as the variables which explain the most variance in the dataset. We initially start by selecting 4 clusters:

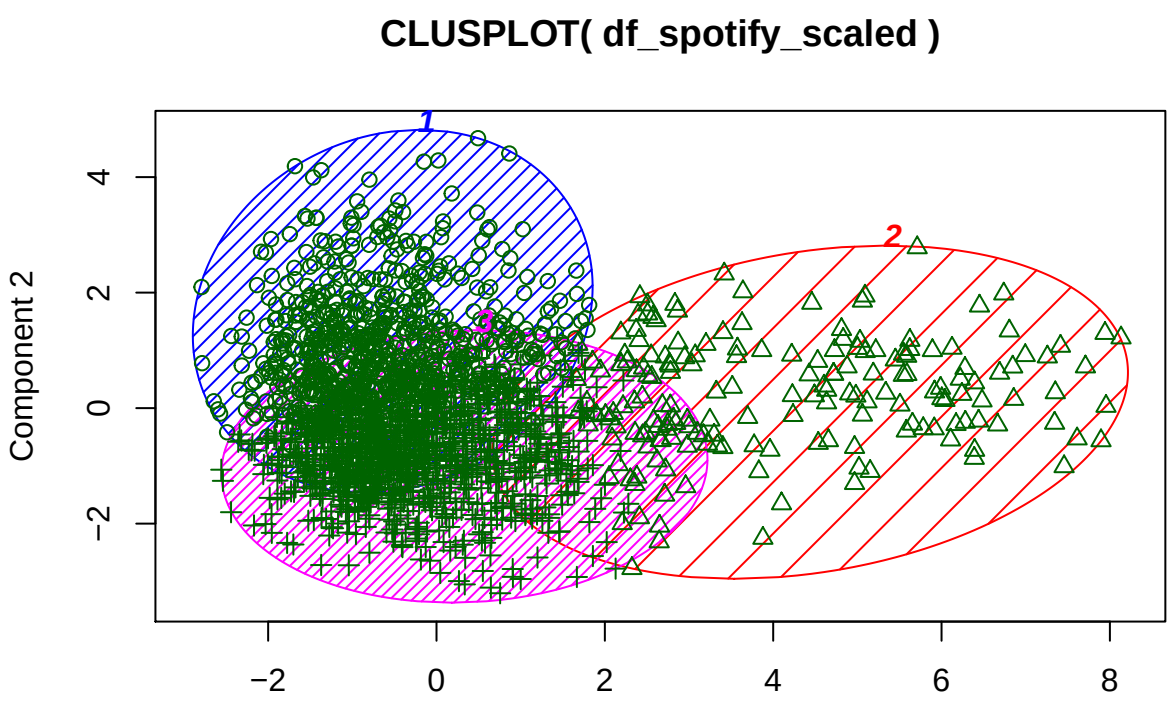
```
# Find cluster solution, K = 4
rsltKmeans <- kmeans(df_spotify_scaled, 4)
# Cluster Plot against 1st 2 principal components
clusplot(df_spotify_scaled, rsltKmeans$cluster,
         color=TRUE, shade=TRUE,
         labels=4, lines=0)
```



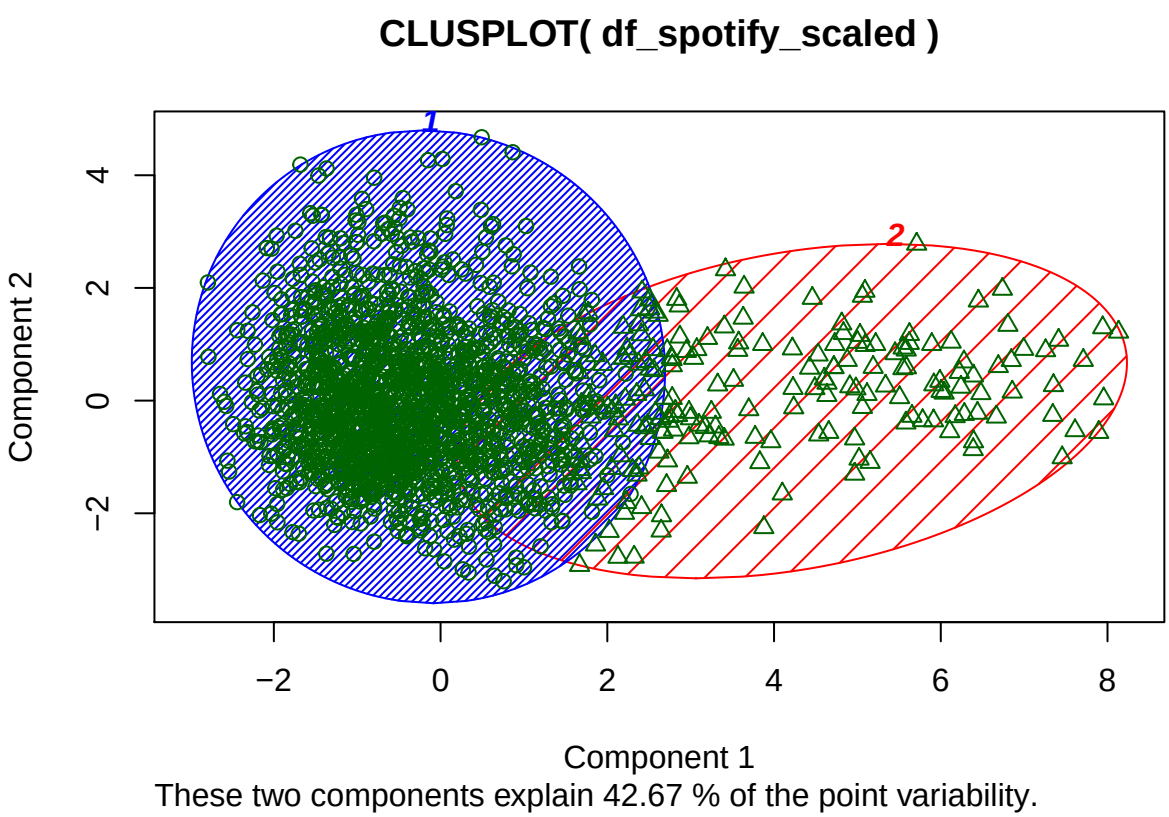
This first plot shows that the 2 selected components explain almost 42.67% of the point variability in the data set. The selected 4 clusters seems to fit the data not very well. Clusters 1, 2 and 4 appear to overlap significantly. For this reason we select to retry with 3 and 2 clusters:

```
# Find cluster solution, K = 3
rsltKmeans <- kmeans(df_spotify_scaled, 3)
```

```
# Cluster Plot against 1st 2 principal components
clusplot(df_spotify_scaled, rsltKmeans$cluster,
         color=TRUE, shade=TRUE,
         labels=4, lines=0)
```



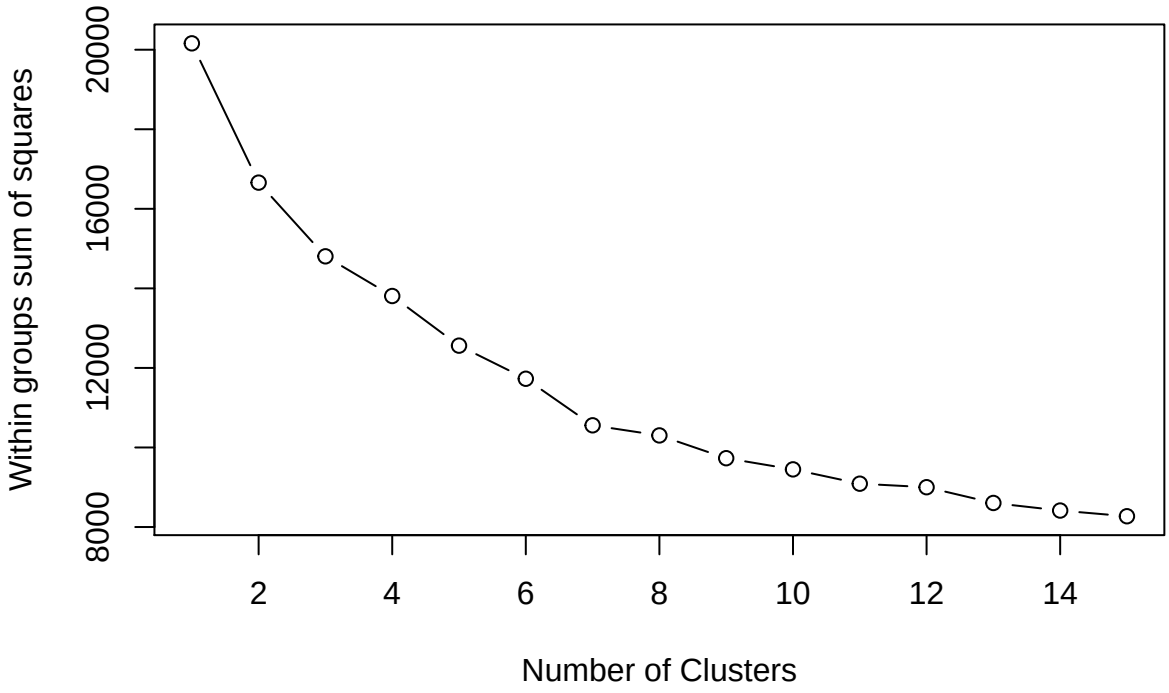

```
# Find cluster solution, K = 2
rsltKmeans <- kmeans(df_spotify_scaled, 2)
# Cluster Plot against 1st 2 principal components
clusplot(df_spotify_scaled, rsltKmeans$cluster,
         color=TRUE, shade=TRUE,
         labels=4, lines=0)
```



Due to the little overlap in the K = 2 graph, we believe that 2 clusters will fit this dataset the best. Although using 3 clusters already improved the clarity of the plot compared to K = 4, there is still quite some overlapping of the blue and pink clusters in the left-middle.

To strengthen our conclusion that we should use 2 clusters we also plot the within cluster variation (WSS) for a variety of clusters sizes (Elbow Method):

```
# The wss variables is initialised
wss <- (nrow(df_spotify_scaled)-1)*sum(apply(df_spotify_scaled,2,var))
# This for loop iterates the number of K clusters from 2 untill 15
for (i in 2:15) wss[i] <- sum(kmeans(df_spotify_scaled,
                                     centers=i)$withinss)
# Plotting the graph
plot(1:15, wss, type="b", xlab="Number of Clusters",
     ylab="Within groups sum of squares")
```

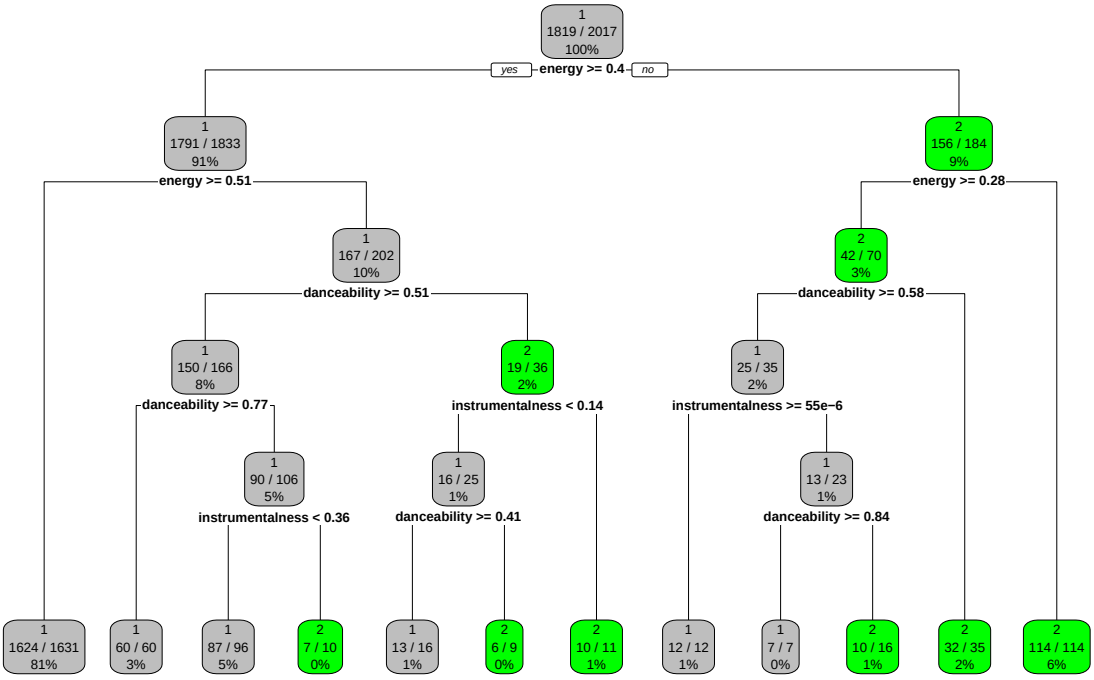


A steep decrease in the plot indicates the right number of clusters as that extra cluster decreases a significant amount of the variation. The steepest decrease can clearly be found for 2 clusters which proves our finding that 2 is the right number of clusters to group the data in.

Explaining the clusters.

To explain the different clusters we have build a decision tree model which predicts cluster membership. The final tree model has been developed after iterating various combinations of independent variables to the model. All variables which were not used in the decision tree (as they apparently did not add any information gain) have been excluded from the final model.

```
#Ensuring that clusters are set to 2.
rsltKmeans <- kmeans(df_spotify_scaled, 2)
# Add the cluster values to the original data set (the target variable)
spotify <- data.frame(spotify, cluster=as.factor(rsltKmeans$cluster))
# A decision tree model to explain Cluster memberships
mdlTree <- cluster ~ loudness + instrumentalness + danceability + energy + liveness + loudness
rsltTree <-rpart(mdlTree,
                 data = spotify,
                 method = "class",
                 parms=list(split="information"))
# Plot the decision tree
rpart.plot(rsltTree,
           box.col= c("grey","green","grey")[rsltTree$frame$yval],
           extra = 102)
```



To explain what songs are in which cluster, we follow the plotted decision tree top-down. It can instantly be noted that the energy variable provides significant information gain as it's used multiple times throughout the model. Songs in cluster 1 have high values for energy whereas songs in cluster 2 have significantly lower values. According to Spotify (2018), songs with higher values fore engird feel "fast, loud and noisy like death metal music while a Bach prelude scores low on the scale".

Music in cluster 2 is also found less 'danceable' according to the decision model. This can be expected as the danceability variable measures similar aspects as the energy variable such as tempo and beat strength. Thus songs in cluster 1 are more upbeat, and danceable. One could assume that these songs are the more mainstream, popular songs and are hence more similar, whereas songs in cluster 2 might be songs of a more 'niche' genre like classical music.

Finally, the instrumentalness variable is used for multiple splits in the tree model but these are not easily interpreted as they are used in the lower areas of the model. To understand this variable, it is useful to know what the values represents. According to Spotify (2018), "the closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks". Not all splits of instrumentalness indicate the same direction (for two splits a lower value leads to cluster 1 whereas for 1 split a lower value leads to cluster 2. For this reason we argue that it is not directly possible to characterize clusters 1 and 2 based on this variable and we stick to using energy and danceability.

References

Spotify, (2018) Get Audio Features for a Track https://developer.spotify.com/web-api/get-audio-features/