Complete task 2 (from scratch)

Gabriel Musker

26/11/2019

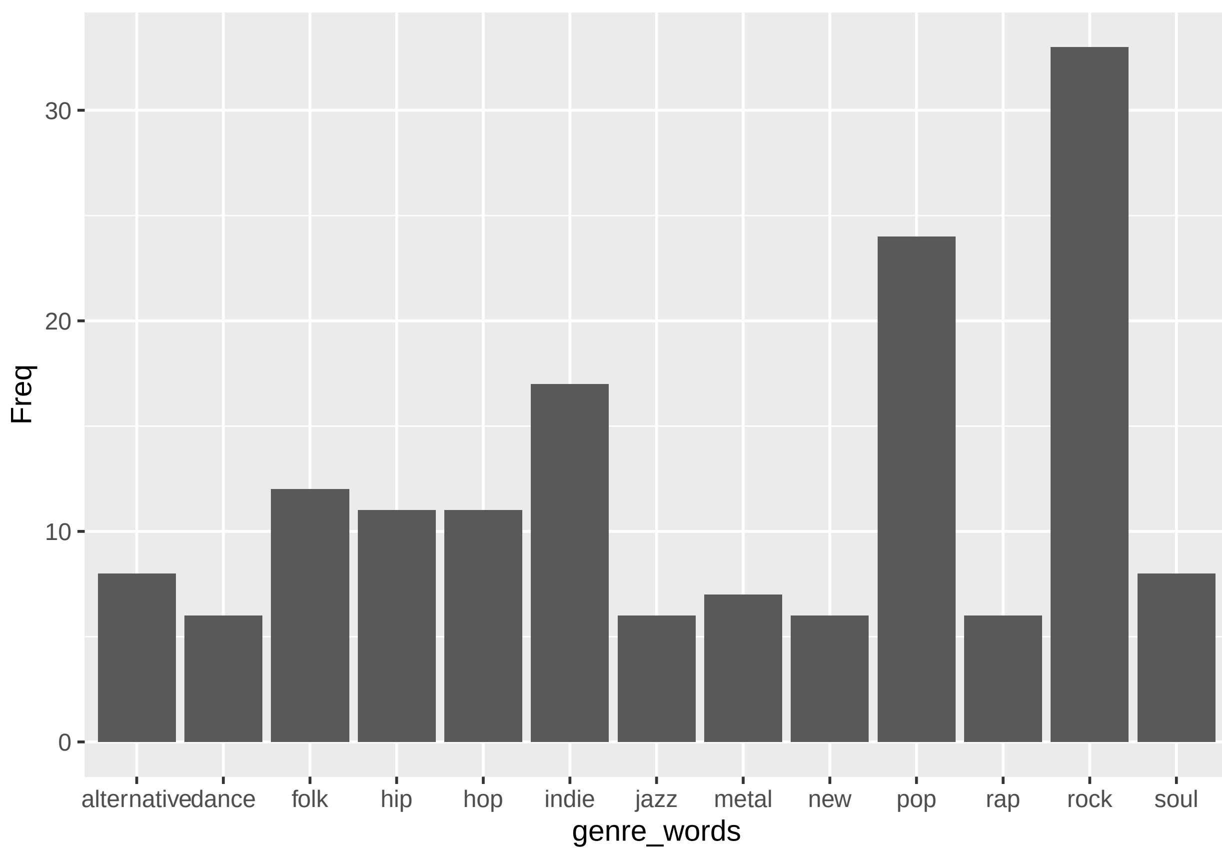
Exploratory Data Analysis

The following chunk of code cleans the data and then splits it into training and testing subsets.

```
spotify <- import %>%
  mutate(AlbumReleaseDate = parse_date_time(AlbumReleaseDate,
                                            orders = c("y", "ym","ymd"))) %>%
  #Old-school grepl method
  mutate(Artist = ifelse(grepl("Beyonc*", Artist), 'Beyonce', Artist)) %>%
  #Tidyverse str_detect method
  mutate(Artist = ifelse(Artist %>%
                         str_detect("Janelle Mon*"), 'Janelle Monae', Artist)) %>%
  mutate(AlbumBestChartPosition = ifelse(AlbumBestChartPosition %>%
                         str_detect("#N/A"), 0, AlbumBestChartPosition)) %>%
  na.omit() %>%
  mutate(id = row_number()) %>%
  mutate(id = as.character(id))
sapply(data, class)

test_data <- subset(spotify,
                    ((AlbumName == "A Girl Called Dusty") |
                     (AlbumName == "Action!") |
                     (AlbumName == "Selling England By The Pound") |
                     (AlbumName == "Carpenters") |
                     (AlbumName == "Ride On") |
                     (AlbumName == "Autoamerican") |
                     (AlbumName == "Selected Ambient Works 85-92") |
                     (AlbumName == "Different Class") |
                     (AlbumName == "O") |
                     (AlbumName == "The Elder Scrolls IV: Oblivion: Original Game Soundtrack") |
                     (AlbumName == "AM") |
                     (AlbumName == "An Awesome Wave")))

training_data <- subset(spotify,
                        ((AlbumName != "A Girl Called Dusty") &
                         (AlbumName != "Action!") &
                         (AlbumName != "Selling England By The Pound") &
                         (AlbumName != "Carpenters") &
                         (AlbumName != "Ride On") &
                         (AlbumName != "Autoamerican") &
                         (AlbumName != "Selected Ambient Works 85-92") &
                         (AlbumName != "Different Class") &
                         (AlbumName != "O") &
                         (AlbumName != "The Elder Scrolls IV: Oblivion: Original Game Soundtrack") &
                         (AlbumName != "AM") &
                         (AlbumName != "An Awesome Wave")))
```

The next code chunk finds the tags that appear more than a given number of times in the training data (under the variable ArtistGenres), produces a table of these words and then prints a bar plot showing their respective frequencies.

```
genres <- mutate(training_data,
                 ArtistGenres = strsplit(ArtistGenres, ","))$ArtistGenres %>%
  unlist() %>%
  unique() %>%
  na.omit()

split_genres <- NULL

for(i in 1:length(genres)) {
  split_genres[i] <- strsplit(genres[i], " ")
}

genre_words <- unlist(split_genres)

cutoff_freq <- 5

most_frequent_words <- subset(as.data.frame(table(genre_words)), (Freq > cutoff_freq))

genre_freq_plot <- most_frequent_words %>%
  ggplot(aes(x=genre_words, y=Freq)) + geom_col()

print(genre_freq_plot)
```



Clustering

The following code creates a short function defined as follows: $f : A \subset \mathbb{R} \to [0,1]$, $f(x) = \frac{x-\min(A)}{\max(A)-\min(A)}$. This will be used later in the code.

```
standardise <- function(x){(x-min(x))/(max(x)-min(x))}
```

This code chunk keeps only the relevant variables for clustering and normalises the values of each variable.

```
clustering_training_data <- training_data[c("TrackDuration", "TrackDanceability",
                                            "TrackEnergy", "TrackKey", "TrackLoudness",
                                            "TrackSpeechiness", "TrackAcousticness",
                                            "TrackInstrumentalness", "TrackLiveness",
                                            "TrackValence", "TrackTempo")] %>%
  mutate(TrackDuration = standardise(TrackDuration)) %>%
  mutate(TrackKey = standardise(TrackKey)) %>%
  mutate(TrackLoudness = standardise(TrackLoudness)) %>%
  mutate(TrackTempo = standardise(TrackTempo))
```

This code produces a matrix of agglomerative coeffiecients for 4 different linkage methods and 6 different distance metrics to use with the hierarchical agglomerative clustering method. It takes ages to run so this chunk has been included but does not evaluate; the evaluated table is included in the technical appendix .zip file.

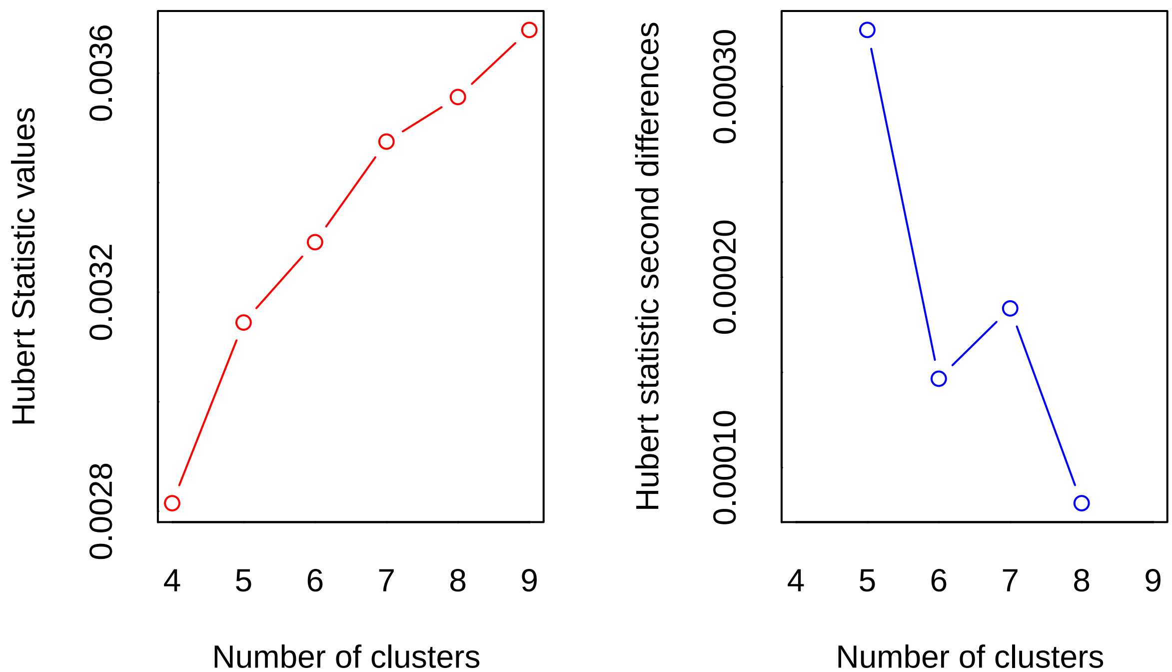
```
# matrix of methods to compare
m <- c( "average", "single", "complete", "ward")
names(m) <- c( "average", "single", "complete", "ward")

distances <- c("euclidean", "maximum", "manhattan", "canberra",
               "binary", "minkowski")
names(distances) <- c("euclidean", "maximum", "manhattan",
                      "canberra", "binary", "minkowski")

clust_comps <- matrix(nrow = length(distances), ncol = length(m),
                      dimnames = list(distances,m))

# function to compute coefficient
ac <- function(distance, linkage) {
  dista <- dist(clustering_training_data, method = distance)
  agnes(dista, method = linkage)$ac
}

for(i in 1:length(distances)) {
  for(j in 1:length(m)) {
    clust_comps[i,j] <- ac(distances[i], m[j])
  }
}
```

The training data is then clustered using the following line of code.

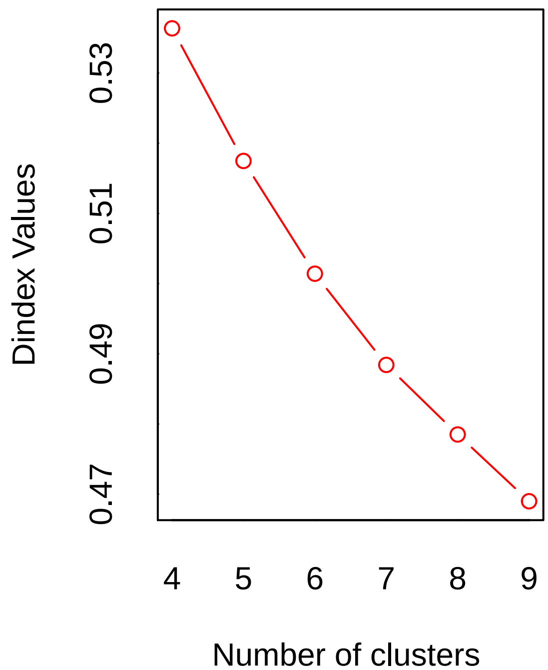
```
clustered <- agnes(dist(clustering_training_data, method = "euclidian"),
                   diss=TRUE, method = "ward")

# add cluster labels to the training data

training_data$cluster <- cutree(clustered, k=5)
```

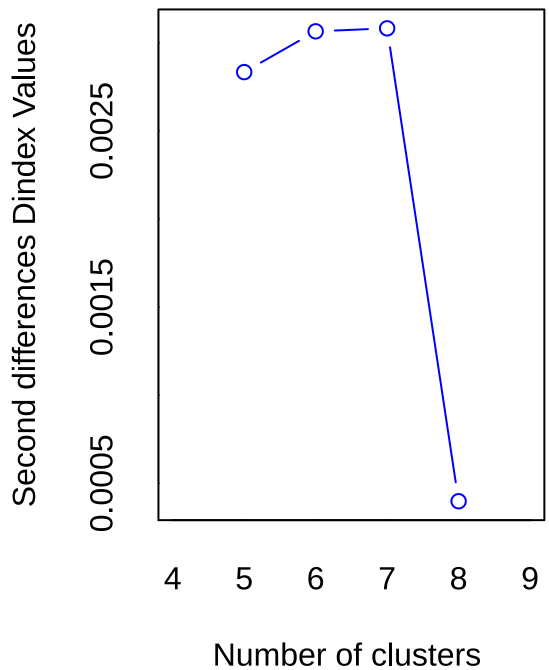
Then the NbClust() function is used to find the best number of clusters for the clustered data.

```
fviz_nbclust(NbClust(clustering_training_data, distance="euclidean",
                                min.nc=4, max.nc=9, method="ward.D2", index="all"))
```



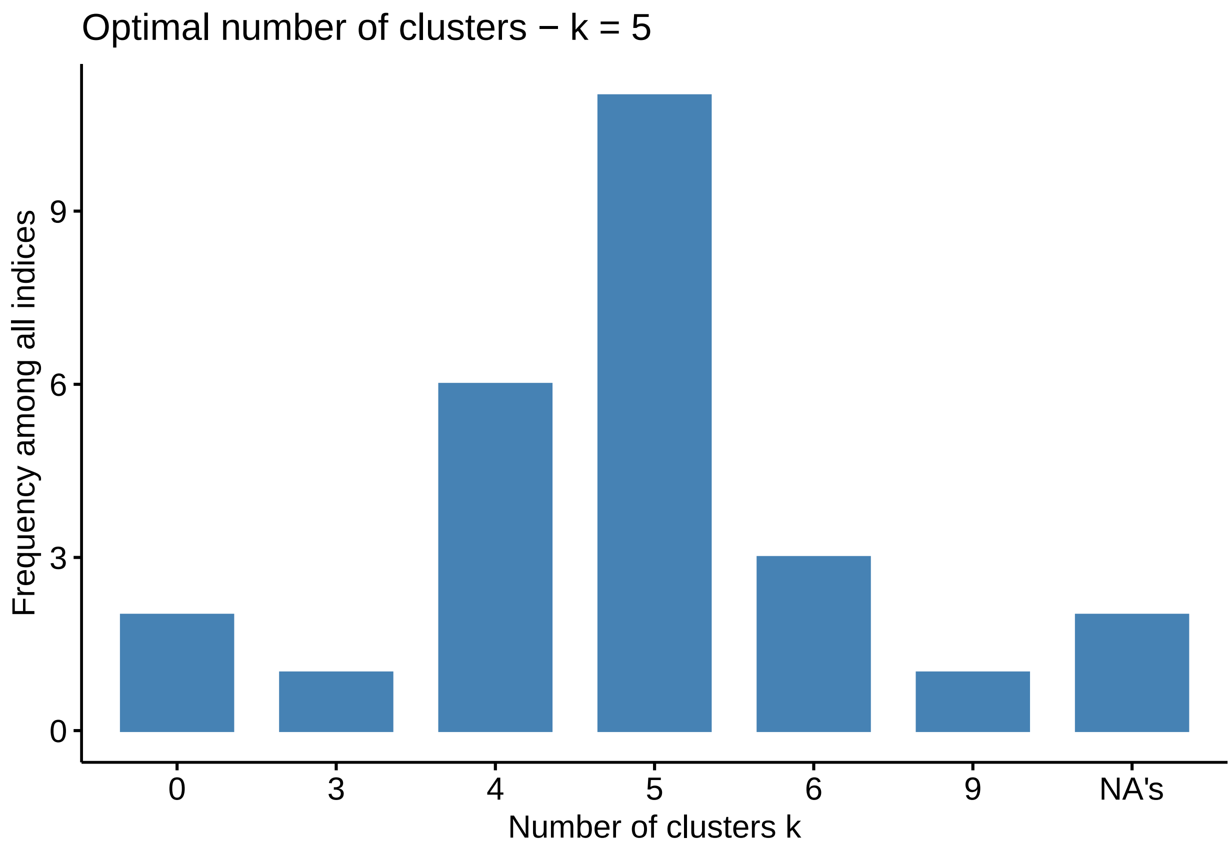

```
## *** : The Hubert index is a graphical method of determining the number of clusters.
##                 In the plot of Hubert index, we seek a significant knee that corresponds to a
##                 significant increase of the value of the measure i.e the significant peak in Hubert
##                 index second differences plot.
##
```






```
## *** : The D index is a graphical method of determining the number of clusters.
##                 In the plot of D index, we seek a significant knee (the significant peak in Dindex
##                 second differences plot) that corresponds to a significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 6 proposed 4 as the best number of clusters
## * 11 proposed 5 as the best number of clusters
## * 3 proposed 6 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  5
##
##
## *******************************************************************
## Among all indices:
## ===================
## * 2 proposed  0 as the best number of clusters
## * 1 proposed  3 as the best number of clusters
## * 6 proposed  4 as the best number of clusters
## * 11 proposed  5 as the best number of clusters
## * 3 proposed  6 as the best number of clusters
## * 1 proposed  9 as the best number of clusters
```

```
## * 2 proposed  NA's as the best number of clusters
##
## Conclusion
## =========================
## * According to the majority rule, the best number of clusters is  5 .
```



The code below produces the dendrogram found in the technical appendix .zip file but is not evaluated in this markdown file for time reasons.

```
fviz_dend(clustered, k=5, show_labels = FALSE)
```

Classification

The code below filters the useful information for the test and training data sets and separates the training data's true labels into another variable.

```
classification_training_data <- training_data[c("TrackName", "TrackDuration", "TrackDanceability",
                                            "TrackEnergy", "TrackKey", "TrackLoudness",
                                            "TrackSpeechiness", "TrackAcousticness",
                                            "TrackInstrumentalness", "TrackLiveness",
                                            "TrackValence", "TrackTempo")] %>%
  mutate(TrackDuration = standardise(TrackDuration)) %>%
  mutate(TrackKey = standardise(TrackKey)) %>%
  mutate(TrackLoudness = standardise(TrackLoudness)) %>%
  mutate(TrackTempo = standardise(TrackTempo))
```

```r
classification_test_data <- test_data[c("TrackName", "TrackDuration", "TrackDanceability",
                                         "TrackEnergy", "TrackKey", "TrackLoudness",
                                         "TrackSpeechiness", "TrackAcousticness",
                                         "TrackInstrumentalness", "TrackLiveness",
                                         "TrackValence", "TrackTempo")] %>%
  mutate(TrackDuration = standardise(TrackDuration)) %>%
  mutate(TrackKey = standardise(TrackKey)) %>%
  mutate(TrackLoudness = standardise(TrackLoudness)) %>%
  mutate(TrackTempo = standardise(TrackTempo))

true_classifications <- training_data$cluster
```

The following function simply classifies a song using the k-Nearest Neighbours algorithm and returns the number cluster it should be in.

```r
# use k value of sqrt(2375) = ~49 by professional convention

# assume song is a name of a track from the classification test data

classify_song <- function(song_name, train_data, labels, K) {

  song <- subset(classification_test_data, TrackName==song_name)
  song <- song[,-1]

  return(knn(train_data, song, labels, K))

}
```

Finally, the completed function takes a song from the test data set, finds the distances to all the other songs in the same cluster, replaces each distance with a sample from the normal distribution (where the mean is the distance and the variance is given) and then returns the song with the smalled so-called "randomised distance" from the input song.

```r
# let exploration be the variance of the sampling distribution

recommend_new_song <- function(song_name, exploration=0.5) {

  # find the cluster that the new track is in

  clusternumber <- classify_song(song_name, classification_training_data[,-1],
                                 true_classifications, 49)

  song <- (subset(classification_test_data, TrackName==song_name))[,-1]

  tracks_in_cluster <- subset(data.frame(classification_training_data,
                                         true_classifications),
                              true_classifications==clusternumber)

  # calculate the euclidean distance between the new song and every other song in the cluster

  distances <- data.frame((1:dim(tracks_in_cluster[,-1])[1]),
                          rep(0, dim(tracks_in_cluster[,-1])[1]))

  for(i in 1:(dim(tracks_in_cluster[,-1])[1])) {
```

```r
    distances[i,2] <- dist(rbind(song, tracks_in_cluster[i,c(-1,-13)]), method = "euclidean")

  }

  colnames(distances) <- c("TrackID", "RandomisedDist")

  # replace distance value with a normally sampled random number based on distance

  for(i in 1:(dim(tracks_in_cluster[,-1])[1])) {

    distances[i,2] <- distances[i,2]*abs(rnorm(1, 1, exploration))

  }

  # sort the randomised distances

  distances <- distances[order(distances$RandomisedDist),]

  # return the name of the song with the smallest randomised distance

  nearest_song <- tracks_in_cluster[distances[1,1],]

  return(nearest_song$TrackName)
}
```