CLUSTERING

AARUSHI PANDEY [1] BRANDON RUNYON [2]
ZACHARY CANOOT [3] GRAY SIMPSON [4]

09 OCTOBER, 2022

WHAT IS OUR DATA?

We like to think that a lot of people are going to use this database! When it comes to clustering, the first use case is automatic recommendations for apps like Netflix or Spotify. Our data, Spotify most unpopular songs (https://www.kaggle.com/datasets/estienneggx/spotify-unpopular-songs), details different characteristics about 10,000+ of the top most disliked songs on Spotify. Each track is an observation with characteristics about the music:

- Danceability (In a range from 0 to 1)
- Energy (In a range from 0 to 1)
- The Key (Integers mapping to pitches the 12 pitches)
- The Mode (Major and Minor although it be crazy if this could identify the likability of modes like mixolydian)
- Loudness (Decimal in floating point) * Speechiness (In a range from 0 to 1)
- Acousticness (In a range from 0 to 1) * Instrumentalness (In a range from 0 to 1)
- Liveness (Presence of audience in a range from 0 to 1)
- Valence (https://en.wikipedia.org/wiki/Valence_(psychology)) (Positiveness from 0 to 1)

As well as more basic statistics, like:

- Tempo (BPM floating point)
- Duration (Integer milliseconds)
- Explicitness (True or False)
- Popularity (I don't know how Spotify measures this integer, but 0 to 18)

And of course, the name of the track, it's id, and the artist. The beauty of all this data is that it can all be scraped using the Spotify API, so any work with this data can be extended to make custom tools that interact with Spotify. If we apply clustering to this data, we will be able to form some groups of within the data and understand what genres might be unpopular on Spotify.

We even have genre data that we can later use to analyze whether the unsupervised clustering is getting close to the genres we saw. The current hypothesis is that that is very unlikely due to the complexity of music, but we shall see!

EXPLORING OUR DATA

First we must load it in!

```
spotify <- read.csv("data/unpopular_songs.csv")
# Saving this so I don't have to convert factors later
spotify_unedited <- spotify
head(spotify)
```

```
##   danceability energy key loudness mode speechiness acousticness
## 1        0.530  0.770   4   -6.633    0      0.0389        0.284
## 2        0.565  0.730   1   -6.063    1      0.0730        0.365
## 3        0.427  0.546   4   -8.727    1      0.0849        0.539
## 4        0.421  0.531   7   -5.516    1      0.0262        0.706
## 5        0.537  0.804   8   -7.378    0      0.1570        0.379
## 6        0.710  0.621   9   -7.879    0      0.0329        0.405
##   instrumentalness liveness valence   tempo duration_ms explicit
popularity
## 1         0.501000    0.744   0.623 120.144      225696    False
2
## 2         0.000000    0.237   0.511 130.026      158093    False
2
## 3         0.015200    0.368   0.435  78.345      167262    False
2
## 4         0.000208    0.110   0.383  85.080      236832    False
2
## 5         0.000489    0.323   0.543 139.950      239400    False
2
## 6         0.001900    0.103   0.546 125.985      194560    False
2
##        track_name track_artist               track_id
## 1      No Regrets James Reeder 6f2c4a9lNx8aowZJngv7cJ
## 2       Wild Life James Reeder 3fTs52jsDzSuVLsifxNKO8
## 3           Fangs James Reeder 6NPafqavrv0icaIHMQnXDy
## 4     Afterburner James Reeder 3vGmhxveURgmlZStvo0uc1
## 5 Hellfire Rising James Reeder 4O2qRbfCHzMMgfbw9DBdGf
## 6       Hurricane James Reeder 1Tu9d0uA2ipK3s8EddNfl9
```

I would load in the genre data, but it is so incomplete, I don't think that we would be able to compare it to our results

Looking good, although we would want to categorize the enumerated attributes key and mode, as well as factor explicitness.

```
# Make qualitative attributes into factors
spotify$key <- as.factor(spotify$key)
spotify$mode <- as.factor(spotify$mode)
spotify$explicit <- as.factor(spotify$explicit)

# Label the factors that are just enumerators
keys <- c("C", "C#", "D", "D#", "E", "F", "F#", "G", "G#", "A", "A#",
        "B")

modes <- c("Major", "Minor")
levels(spotify$mode) <- as.factor(modes)
levels(spotify$key) <- as.factor(keys)
str(spotify)
```

```
## 'data.frame':    10877 obs. of  17 variables:
##  $ danceability    : num  0.53 0.565 0.427 0.421 0.537 0.71 0.419
0.565 0.547 0.533 ...
##  $ energy          : num  0.77 0.73 0.546 0.531 0.804 0.621 0.821
0.624 0.56 0.785 ...
##  $ key             : Factor w/ 12 levels "C","C#","D","D#",..: 5 2
5 8 9 10 12 2 1 6 ...
##  $ loudness        : num  -6.63 -6.06 -8.73 -5.52 -7.38 ...
##  $ mode            : Factor w/ 2 levels "Major","Minor": 1 2 2 2 1
1 1 2 2 2 ...
##  $ speechiness     : num  0.0389 0.073 0.0849 0.0262 0.157 0.0329
0.0431 0.0351 0.051 0.0481 ...
##  $ acousticness    : num  0.284 0.365 0.539 0.706 0.379 0.405
0.0137 0.00442 0.551 0.591 ...
##  $ instrumentalness: num  0.501 0 0.0152 0.000208 0.000489 0.0019
0.00365 0.221 0.179 0 ...
##  $ liveness        : num  0.744 0.237 0.368 0.11 0.323 0.103 0.127
0.108 0.137 0.162 ...
##  $ valence         : num  0.623 0.511 0.435 0.383 0.543 0.546 0.343
0.655 0.354 0.521 ...
##  $ tempo           : num  120.1 130 78.3 85.1 139.9 ...
##  $ duration_ms     : int  225696 158093 167262 236832 239400 194560
195288 211043 182184 120936 ...
##  $ explicit        : Factor w/ 2 levels "False","True": 1 1 1 1 1 1
1 1 1 1 ...
##  $ popularity      : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ track_name      : chr  "No Regrets" "Wild Life" "Fangs"
"Afterburner" ...
##  $ track_artist    : chr  "James Reeder" "James Reeder" "James
Reeder" "James Reeder" ...
##  $ track_id        : chr  "6f2c4a9lNx8aowZJngv7cJ"
"3fTs52jsDzSuVLsifxNKO8" "6NPafqavrv0icaIHMQnXDy"
"3vGmhxveURgmlZStvo0uc1" ...
```

Now our data is easier to read! Note that I labeled the data major and minor, while we don't actually know if 0 or 1 equals major or minor. I suspect a strong bias towards the first factor being major. In the case of our clustering results this doesn't matter!

To get a rough overview:

```
summary(spotify)
```

```
##   danceability        energy                 key         loudness      
##  Min.   :0.0000   Min.   :0.0000203   C#     :1286   Min.   
:-51.808  
##  1st Qu.:0.4420   1st Qu.:0.3790000   C      :1255   1st 
Qu.:-13.796  
##  Median :0.6020   Median :0.5690000   G      :1237   Median : 
-9.450  
##  Mean   :0.5725   Mean   :0.5497713   A      :1102   Mean   
:-11.359  
##  3rd Qu.:0.7300   3rd Qu.:0.7450000   D      :1097   3rd Qu.: 
-6.726  
##  Max.   :0.9860   Max.   :1.0000000   B      : 834   Max.   :  
3.108  
##                                       (Other):4066                   
##     mode       speechiness      acousticness    instrumentalness  
##  Major:3905   Min.   :0.0000   Min.   :0.0000   Min.   :0.000000  
##  Minor:6972   1st Qu.:0.0384   1st Qu.:0.0365   1st Qu.:0.000000  
##               Median :0.0589   Median :0.2330   Median :0.000133  
##               Mean   :0.1380   Mean   :0.3542   Mean   :0.232943  
##               3rd Qu.:0.1880   3rd Qu.:0.6570   3rd Qu.:0.517000  
##               Max.   :0.9620   Max.   :0.9960   Max.   :1.000000  
##                                                                   
##     liveness         valence           tempo        duration_ms     
##  Min.   :0.0000   Min.   :0.0000   Min.   :  0.0   Min.   :   4693  
##  1st Qu.:0.0993   1st Qu.:0.2380   1st Qu.: 93.0   1st Qu.: 151152  
##  Median :0.1290   Median :0.4680   Median :117.1   Median : 197522  
##  Mean   :0.2121   Mean   :0.4646   Mean   :117.8   Mean   : 205578  
##  3rd Qu.:0.2680   3rd Qu.:0.6850   3rd Qu.:138.9   3rd Qu.: 244428  
##  Max.   :0.9990   Max.   :0.9950   Max.   :239.5   Max.   :3637277  
##                                                                     
##   explicit      popularity      track_name        track_artist      
##  False:7945   Min.   : 0.000   Length:10877       Length:10877      
##  True :2932   1st Qu.: 1.000   Class :character   Class :character  
##               Median : 2.000   Mode  :character   Mode  :character  
##               Mean   : 3.079                                        
##               3rd Qu.: 3.000                                        
##               Max.   :18.000                                        
##                                                                     
##    track_id        
##  Length:10877      
##  Class :character  
##  Mode  :character  
##                    
##                    
##                    
##                    
```

We can see from just the basic metrics for each attribute - Danceability and energy's mean approaches a .5, which suggests a weak correlation to the unpopularity of the data set.

- The mean for tempo is 117, which approaches the common 120 bpm
- The songs are not very acoustic, nor are they *speechy* (~.35 and ~.14 respectively), nor are they very instrumental. That hints they might have pretty heavily virtual sounds.
- All of the decibel levels are negative, which lead me to realize that volume normalization plays a role. Reading [spotify's normalization summary (https://artists.spotify.com/en/help/article/loudness-normalization)](https://artists.spotify.com/en/help/article/loudness-normalization) helped me realize how it worked. The idea is to aim for -14 db on this scale, so the fact that the mean is -14 shows there might be some bad mastering for these songs.

Since we aren't trying to predict anything outright, just analyzing the data, lets clean up the data and get going!

```
# There are no na values!
sum(sapply(spotify, is.na))
```

```
## [1] 0
```

```
# Now we want to remove
```

CLUSTERING ALGORITHMS

I saw an a quote (https://datascience.stackexchange.com/questions/22/k-means-clustering-for-mixed-numeric-and-categorical-data) on why kMeans shouldn't use categorical data: "The fact a snake possesses neither wheels nor legs allows us to say nothing about the relative value of wheels and legs". There is no numerical *distancing* we can really use with categorical data in clustering. Of course the Mode, Key, and Explicitness could still be useful in the clustering of a musical database. If we were to take this categorical data into account while expressing them numerically we would get a result, and it *may* be useful because each of these values comes from a *range* of possible values.

We can remove the factors for one data set, and convert them to integers for another data set and check results. We also can then scale the data using the scale function. Now the mean of each attribute is 0, and each factor is easily comparable to all other factors.

```
# DF of just numeric values
spotify_num <- spotify[sapply(spotify, is.numeric)]
# DF with categorical data, with true false converted to ints
# from 0 to 1 (Other then names and artist)
spotify_fact <- spotify_unedited
spotify_fact$explicit <-
       as.numeric(as.factor(spotify_fact$explicit))-1

spotify_fact <- spotify_fact[sapply(spotify_fact, is.numeric)]

# Now we scale that categorical data using the scale function
scaled_num <- as.data.frame(scale(spotify_num))
scaled_fact <- as.data.frame(scale(spotify_fact))

summary(scaled_num)
```

```
##   danceability         energy            loudness
speechiness
##  Min.   :-2.8133   Min.   :-2.13522   Min.   :-5.9582   Min.
:-0.8618
##  1st Qu.:-0.6414   1st Qu.:-0.66327   1st Qu.:-0.3590   1st
Qu.:-0.6219
##  Median : 0.1449   Median : 0.07468   Median : 0.2812   Median
:-0.4939
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000   Mean   :
0.0000
##  3rd Qu.: 0.7739   3rd Qu.: 0.75826   3rd Qu.: 0.6825   3rd Qu.:
0.3126
##  Max.   : 2.0318   Max.   : 1.74867   Max.   : 2.1310   Max.   :
5.1474
##   acousticness     instrumentalness     liveness          valence
##  Min.   :-1.0389   Min.   :-0.6312   Min.   :-1.1147   Min.
:-1.70082
##  1st Qu.:-0.9318   1st Qu.:-0.6312   1st Qu.:-0.5929   1st
Qu.:-0.82950
##  Median :-0.3554   Median :-0.6309   Median :-0.4369   Median :
0.01253
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   :
0.00000
##  3rd Qu.: 0.8883   3rd Qu.: 0.7697   3rd Qu.: 0.2935   3rd Qu.:
0.80696
##  Max.   : 1.8827   Max.   : 2.0785   Max.   : 4.1348   Max.   :
1.94187
##      tempo           duration_ms         popularity
##  Min.   :-3.77730   Min.   :-1.88160   Min.   :-0.76840
##  1st Qu.:-0.79605   1st Qu.:-0.50978   1st Qu.:-0.51883
##  Median :-0.02201   Median :-0.07546   Median :-0.26927
##  Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.00000
##  3rd Qu.: 0.67663   3rd Qu.: 0.36389   3rd Qu.:-0.01971
##  Max.   : 3.90056   Max.   :32.14312   Max.   : 3.72372
```

```
summary(scaled_fact)
```

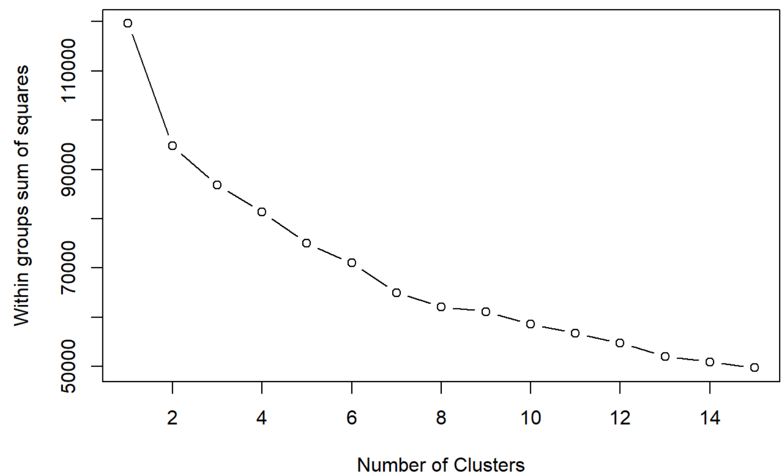
```
##   danceability         energy              key
loudness
##  Min.   :-2.8133   Min.   :-2.13522   Min.   :-1.44278   Min.
:-5.9582
##  1st Qu.:-0.6414   1st Qu.:-0.66327   1st Qu.:-0.89034   1st
Qu.:-0.3590
##  Median : 0.1449   Median : 0.07468   Median :-0.06168   Median :
0.2812
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   :
0.0000
##  3rd Qu.: 0.7739   3rd Qu.: 0.75826   3rd Qu.: 1.04319   3rd Qu.:
0.6825
##  Max.   : 2.0318   Max.   : 1.74867   Max.   : 1.59563   Max.   :
2.1310
##       mode          speechiness       acousticness
instrumentalness
##  Min.   :-1.3361   Min.   :-0.8618   Min.   :-1.0389   Min.
:-0.6312
##  1st Qu.:-1.3361   1st Qu.:-0.6219   1st Qu.:-0.9318   1st
Qu.:-0.6312
##  Median : 0.7484   Median :-0.4939   Median :-0.3554   Median
:-0.6309
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   :
0.0000
##  3rd Qu.: 0.7484   3rd Qu.: 0.3126   3rd Qu.: 0.8883   3rd Qu.:
0.7697
##  Max.   : 0.7484   Max.   : 5.1474   Max.   : 1.8827   Max.   :
2.0785
##     liveness          valence             tempo
duration_ms
##  Min.   :-1.1147   Min.   :-1.70082   Min.   :-3.77730   Min.
:-1.88160
##  1st Qu.:-0.5929   1st Qu.:-0.82950   1st Qu.:-0.79605   1st
Qu.:-0.50978
##  Median :-0.4369   Median : 0.01253   Median :-0.02201   Median
:-0.07546
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   :
0.00000
##  3rd Qu.: 0.2935   3rd Qu.: 0.80696   3rd Qu.: 0.67663   3rd Qu.:
0.36389
##  Max.   : 4.1348   Max.   : 1.94187   Max.   : 3.90056   Max.
:32.14312
##     explicit         popularity
##  Min.   :-0.6075   Min.   :-0.76840
##  1st Qu.:-0.6075   1st Qu.:-0.51883
##  Median :-0.6075   Median :-0.26927
##  Mean   : 0.0000   Mean   : 0.00000
##  3rd Qu.: 1.6461   3rd Qu.:-0.01971
##  Max.   : 1.6461   Max.   : 3.72372
```

KMEANS CLUSTERING

Using the examples from the wine data experiment in class, we will plot the size of the clusters in the data as well as analyze the usefulness of each number of clusters.

```
wsplot <- function(data, nc=15, seed=1234){
  wss <- (nrow(data)-1)*sum(apply(data,2,var))
  wss
  for (i in 2:nc){
    set.seed(seed)
    wss[i] <- sum(kmeans(data,centers=i)$withinss)
  }
  plot(1:nc, wss, type="b", xlab="Number of Clusters",
       ylab="Within groups sum of squares")
}
wsplot(scaled_num)
```

```
## Warning: did not converge in 10 iterations
```

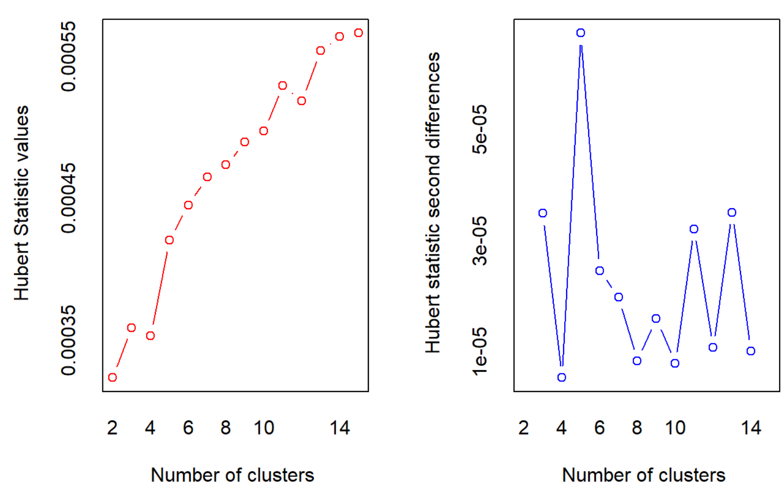


If the analysis function doesn't work with a small sample size, then we know this data probably isn't going to be too easy to cluster.

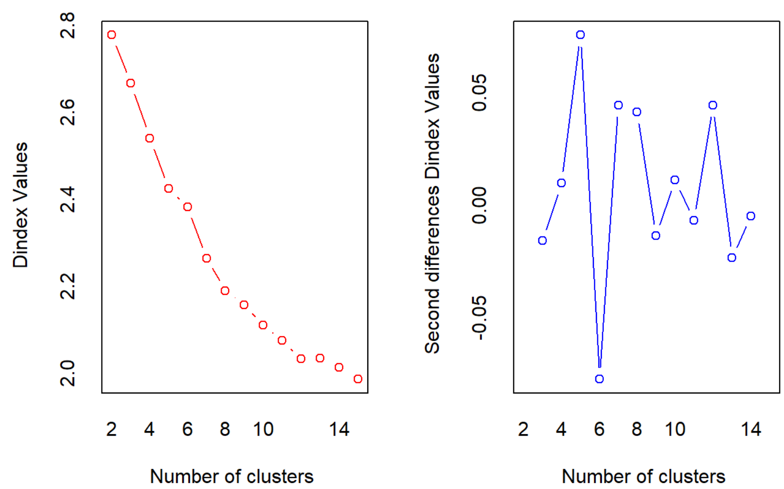
```
library(NbClust)
set.seed(1234)
analyze <- function(data) {
  nc <- NbClust(data, min.nc=2, max.nc=15, method="kmeans")
  table(nc$Best.n[1,])
  barplot(table(nc$Best.n[1,]),
        xlab="Number of Clusters", ylab="Number of Criteria",
        main="Number of Clusters Chosen by 26 Criteria")
}
# Create a cluster analysis object for both data frames from a sample
        space of

# the original very large data set
sample_index <- sample(1:nrow(scaled_num), 1000, replace = FALSE)
sampled_num <- scaled_num[sample_index, ]
# The results of the analysis should be the same for data with
        factors, no need

sampled_fact <- scaled_fact[sample_index, ]
analyze(sampled_num)
```

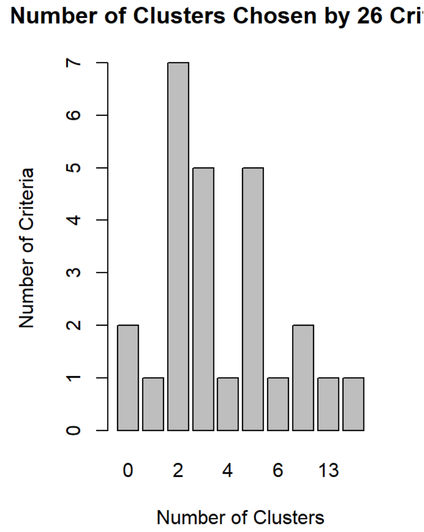


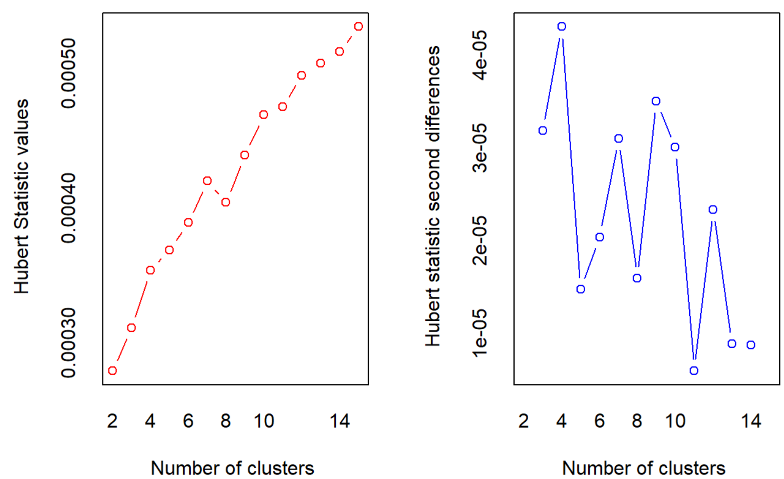
```
## *** : The Hubert index is a graphical method of determining the
number of clusters.
##                 In the plot of Hubert index, we seek a significant
knee that corresponds to a
##                 significant increase of the value of the measure
i.e the significant peak in Hubert
##       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



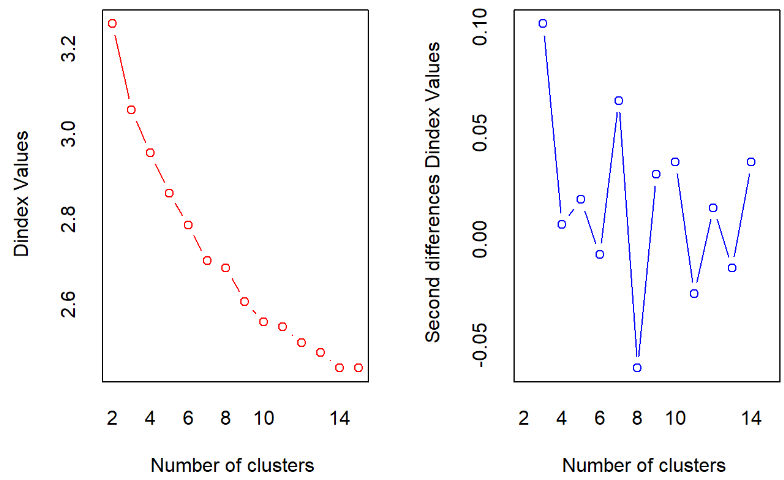

```
## *** : The D index is a graphical method of determining the number
of clusters.
##                 In the plot of D index, we seek a significant knee
(the significant peak in Dindex
##                 second differences plot) that corresponds to a
significant increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 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
## * 5 proposed 5 as the best number of clusters
## * 1 proposed 6 as the best number of clusters
## * 2 proposed 12 as the best number of clusters
## * 1 proposed 13 as the best number of clusters
## * 1 proposed 14 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```

```
analyze(sampled_fact)
```

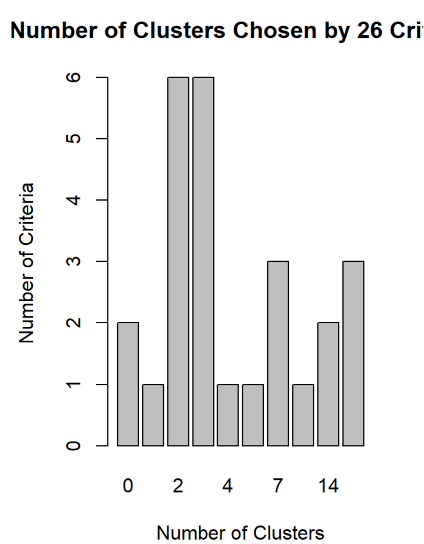





```
## *** : The Hubert index is a graphical method of determining the
number of clusters.
##                  In the plot of Hubert index, we seek a significant
knee that corresponds to a
##                  significant increase of the value of the measure
i.e the significant peak in Hubert
##                  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 2 as the best number of clusters
## * 6 proposed 3 as the best number of clusters
## * 1 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 3 proposed 7 as the best number of clusters
## * 1 proposed 12 as the best number of clusters
## * 2 proposed 14 as the best number of clusters
## * 3 proposed 15 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  2
##
##
## *******************************************************************
```



I just sampled 1000 attributes in the data set, but it appears that 2 is a good number of clusters for general analysis of the data when the categorical data is not included. But if we are factoring in categorical data, 3 clusters could yeild some good results.

Now that we know what might be the most useful number of clusters, we can analyze them!

```
fit.km1 <- kmeans(scaled_num, 2, nstart=25)
fit.km1$size
```

```
## [1] 8160 2717
```

```
fit.km1$centers
```

```
##   danceability     energy   loudness speechiness acousticness
instrumentalness
## 1    0.2903623  0.3338186  0.4104764   0.1460452   -0.3615074
-0.3333411
## 2   -0.8720488 -1.0025615 -1.2327888  -0.4386194    1.0857197
1.0011275
##      liveness    valence      tempo duration_ms popularity
## 1 -0.02831832  0.3037025  0.1445165  0.09827227  0.1004234
## 2  0.08504875 -0.9121135 -0.4340283 -0.29514233 -0.3016030
```

```
fit.km2 <- kmeans(scaled_fact, 3, nstart=25)
fit.km2$size
```

```
## [1] 3074 5313 2490
```

```
fit.km2$centers
```

```
##   danceability     energy          key   loudness        mode
speechiness
## 1   0.63394683  0.1764936 -0.008938120  0.2983697 -0.19216755
1.0691341
## 2   0.05790855  0.3841677  0.009957292  0.4471589  0.10453593
-0.4004593
## 3  -0.90619303 -1.0376000 -0.010211772 -1.3224674  0.01418621
-0.4654128
##   acousticness instrumentalness    liveness    valence       tempo
duration_ms
## 1   -0.3635197       -0.5926271 -0.01634684  0.1510907  0.07616036
-0.1491515
## 2   -0.3171093       -0.1666264 -0.02541671  0.3596070  0.17201437
0.2458184
## 3    1.1254061        1.0871574  0.07441332 -0.9538333 -0.46105594
-0.3403782
##     explicit popularity
## 1  1.5089700  0.1933201
## 2 -0.6015182  0.0278737
## 3 -0.5794005 -0.2981361
```

```
aggregate(spotify_num, by=list(cluster=fit.km1$cluster), mean)
```

```
##   cluster danceability    energy   loudness speechiness
acousticness
## 1       1    0.6316077 0.6357191  -8.572414  0.16134469
0.2309199
## 2       2    0.3950555 0.2916432 -19.728225  0.06774763
0.7243006
##   instrumentalness  liveness   valence    tempo duration_ms
popularity
## 1        0.1099271 0.2067482 0.5475350 122.3355    216070.0
3.481373
## 2        0.6023999 0.2283223 0.2154348 104.2886    174067.9
1.870445
```

```
aggregate(spotify_fact, by=list(cluster=fit.km2$cluster), mean)
```

```
##   cluster danceability    energy      key   loudness      mode
speechiness
## 1       1    0.7015277 0.5952128 5.190956  -9.333485 0.5487964
0.30911903
## 2       2    0.5843031 0.6486824 5.259364  -8.323383 0.6911350
0.07385654
## 3       3    0.3881071 0.2826219 5.186345 -20.337036 0.6477912
0.06345835
##   acousticness instrumentalness  liveness   valence    tempo
duration_ms
## 1    0.2302339       0.01424005 0.2090264 0.5058491 120.2033
189654.3
## 2    0.2460559       0.17145152 0.2073004 0.5628054 123.1933
231822.5
## 3    0.7378303       0.63414845 0.2262983 0.2040390 103.4456
169238.3
##      explicit popularity
## 1 0.939167209   3.853611
## 2 0.002635046   3.190664
## 3 0.012449799   1.884337
```

Models of this data are horrendous, so to spare you a visual assault of colors we can skip it

Looking at this data we can draw conclusions on how the different traits of a song can change the popularity. Generally songs are more popular if:

- They are more lyrical (speechiness and instrumentalness)
- They are a bit louder (threshold of -11)
- More electronic (not acoustic) - faster in tempo
- Happier in valence, but not necessarily lively!
- Longer in duration

If we allow seperate further into explicit and not explicit songs, it seems explicit songs are generally more popular in this range of unpopular songs. Explicit songs are also perhaps a bit more danceable, or have a heavier beat.

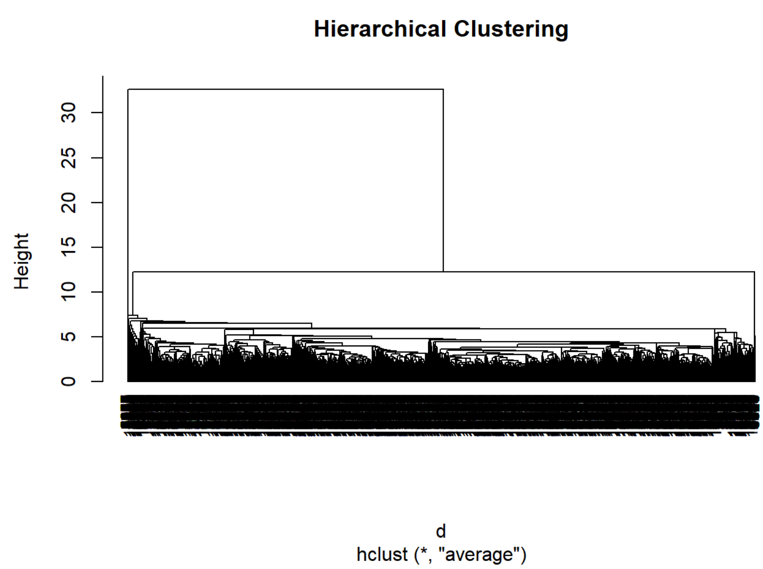
> I want to point out the problem here, I included popularity in the model! The reasoning for that is, well we never would be able to accurately predict popularity anyway, the data is just too varied. What we could do is cluster data into sections to get a vague idea for *types* of unpopular songs. In this case, there are unpopular songs that are super unpopular seem to be a bit less pop-like.

HIERARCHICAL CLUSTERING

Lets look at hierarchical clustering, where we segment the data into trees with each split a division in a particular attribute. We can see above that the split for Explicitness was rather meaningful, helping us create 3 categories of unpopular songs. So from now on I'll use the `spotify_fact` data and `scaled_fact` we have already made above.

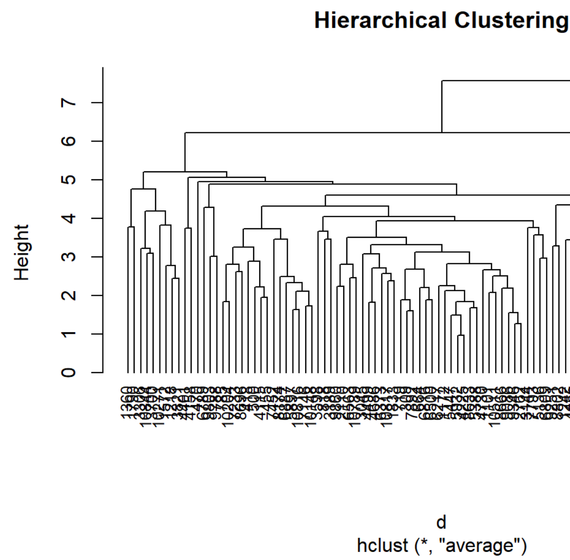
We can

```
d <- dist(scaled_fact)
fit.average <- hclust(d, method="average")
plot(fit.average, hang=-1, cex=.8, main="Hierarchical Clustering")
```



WOoaaaah there! Lets do this on a sample instead and cut it to create actual categories....

```
sample_index <- sample(1:nrow(scaled_num), 100, replace = FALSE)
sampled_fact <- scaled_fact[sample_index, ]
d <- dist(sampled_fact)
fit.average <- hclust(d, method="average")
plot(fit.average, hang=-1, cex=.8, main="Hierarchical Clustering")
```



> This is where I plead. I couldn't get any time off work and am running low on time. While I know that I could do more cutting to get better results, I can clearly see that the data isn't very hierarchical and can stop here without *not* learning anything.

MODEL-BASED CLUSTERING

So, after seeing that our data isn't very hierarchical but we can still see there is a large simple trend in what type of music might be a little less unpopular. Model Based Clustering assumes there is some kind of generating model underneath the data. Our data is a ranking of the least popular 10000+ songs on amazon so theoretically there is a *trend* to the data.

From the article we were given, Mclust is a magical little function that runs

```
# Model Based Clustering
library(mclust)
```

```
## Package 'mclust' version 5.4.10
## Type 'citation("mclust")' for citing this R package in
publications.
```

```
fit <- Mclust(scaled_num)
summary(fit) # display the best model
```

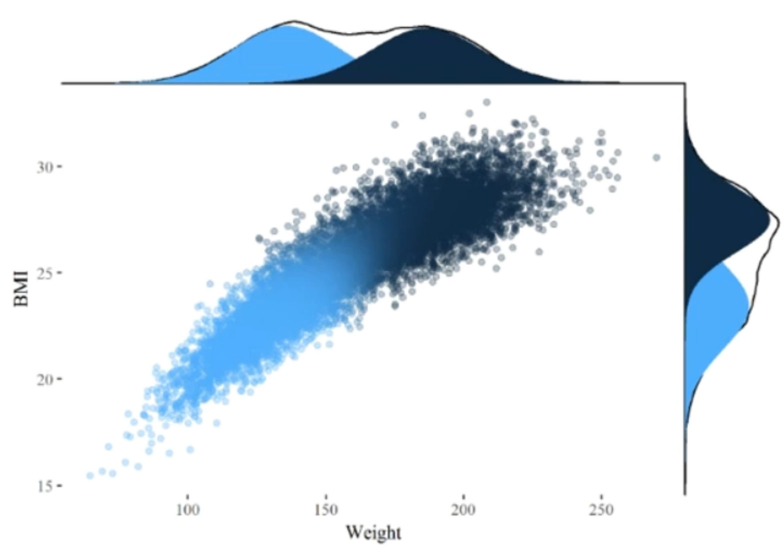
```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation)
model with 5
## components:
##
##  log-likelihood     n  df       BIC       ICL
##       -84974.53 10877 389 -173564.6 -174405.8
##
## Clustering table:
##    1    2    3    4    5
## 4196 1132 2882  845 1822
```

Well this was very unexpected. It actually got much closer to interpretable results. I guess that makes sense considering that the values in the database were generated from some linear trend. It is just that the data is so complex (I mean if we could predict if music was going to be popular we would be very rich).

> Specifically, the Mclust( ) function in the mclust (http://cran.r-project.org/web/packages/mclust/index.html) package selects the optimal model according to BIC for EM initialized by hierarchical clustering for parameterized Gaussian mixture models.

EM, or expectation maximization, found that assuming the model was "VVI (diagonal, varying volume and shape)" was the most likely to produce an accurate model of the data. Then, it preformed a hierarchical clustering with the starting point being clusters that divided the data, with the guidance that the model was diagonal with varying volume and shape. In this case, 5 clusters positioned as if the data was diagonal fit the model the best.

In an example from this video (https://www.youtube.com/watch?v=vC7QF1-JLwI), we can see an example of what this data does:



While this doesn't relate to our data above, it helps explain what our model *is*. The model distribution that model-based clustering found divided the data into clusters that fit a Gaussian distribution underneath the probability curve of the assumed model. Our data in this case, was divided into 5 clusters that fit under the curve.

There are too many parameters here for a visual analysis but we can try and look at the mean values of each of the cluster to get an idea of what the model divided them into:

```
fit$parameters$mean
```

```
##                         [,1]        [,2]        [,3]       [,4]
[,5]
## danceability      0.38413353 -1.03201957 -0.09912722 -0.7441907
0.25323173
## energy            0.18394246  0.23638489 -0.08994919 -1.6287830
0.30213481
## loudness          0.41597447 -0.93742056  0.04825309 -1.6828598
0.32011959
## speechiness       0.51552479  0.07422115 -0.60558018 -0.5462229
-0.04650267
## acousticness     -0.20678989  0.17380350  0.00674281  1.6375366
-0.38152188
## instrumentalness -0.63121297  0.64472157  0.11312693  1.8009940
0.04495505
## liveness         -0.01973904  1.53963135 -0.34407979 -0.5374099
-0.14476118
## valence           0.25728625 -0.74309580  0.12226992 -0.8232509
0.06311542
## tempo             0.06208680 -0.31191718  0.01151583 -0.4466892
0.23323698
## duration_ms      -0.04061797  0.20409416  0.13685768 -0.4922002
-0.02078897
## popularity        0.12035046 -0.47807428 -0.33973389 -0.4162162
0.71695947
```

Looking at the mean of the different clusters, because the original data was already scaled kind of arbitrarily and I couldn't get the aggregation function to work, I am comparing the mean of the scaled values in the different functions

> The goal was learning *something* about the data

Looking at the 5 groups, I am most interested in the 2 clusters that are the max and min of the mean range of popularity. Cluster 1, has the lowest popularity and is characterized by:

- Values lower than -1: energy, loudness
- Values from -1-0: danceability, speechiness, liveness, valence, tempo, duration
- Values greater than 1: Acousticness and Instrumentalness
- Popularity: .816217187

Looks like cluster 1 is slow paced acoustic music

Cluster 4:

- Values from -1-0: Acousticness, liveness, and duration
- Values from 0-1: danceability, energy, loudness, speechiness, instrumentalness, valence, tempo
- Popularity: 0.816217187

This tells me there is a cluster of music that is quite poppy. That is What I would expect!

CONCLUSION AND ANALYSIS

The data itself was very spread on genre with not much overlap as could be seen from the included genre data. However, clustering was still able to reveal trends in what kind of songs were in this lower popularity of music.

- kMeans was a little bit too simple for this data, with the clusters not really able to divide the music by genre. It was quite good in small cluster numbers however, because it could show us how different qualities of music might make the music more popular. The data was very dense, so it could be expected that randomly assigning centroids wouldn't yield a tailored result.

- Hierarchical data was something we fully expected not to work. It would take a neural network to try and get close to our perception of what a sub genre of music is. That is why we only really examined the large deprogram. It clearly separated the music, but in a way that couldn't be more meaningful then whats to come:

- Model-Based clustering feels like magic. The implementation of comparing the likelihoods of a model fitting a given value, and dividing the clusters to maximize probability is very ingenious. The result was the overall diagonal shape being found in the data. Something that I could not have seen in the data. This also meant the 5 clusters found were a bit more purposefully placed along a range in the data. This result was very promising in helping an analyst discover what genres don't work that well on spotify.

The last section there really shows the potential of clustering, as while we didn't gain too much concrete data on the model, I can easily see comparing these results to labels during preparation for a larger prediction algorithm or integration into a larger data set.

---

1. Aarushi's Portfolio (https://github.com/Aarushi-Pandey/Portfolio_ML)↩
2. Brandon's Portfolio (https://github.com/Unicoranium/CS4375)↩
3. Zaiquiri's Portfolio (https://zaiquiriw.github.io/ml-portfolio/)↩
4. Gray's Porfolio (https://ecclysium.github.io/MachineLearning_Portfolio/)↩