DIMENSIONALITY

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

09 OCTOBER, 2022

WHAT IS OUR DATA?

Using the dataset Spotify Unpopular Songs (https://www.kaggle.com/datasets/estienneggx/spotify-unpopular-songs). It contains audio characteristics of many unpopular songs such as perceived intensity, key, decibels, popularity, and more.

Here, we're going to attempt to see if we can manage to find a way to sort songs into general classes (horrible, bad, meh, and passable) based off their popularity scores.

EXPLORING OUR DATA

INITIAL PROCESSING

In this notebook, we will be performing dimensionality reduction to attempt to improve performance and accuracy in kNN regression.

Let's read in the data and take a peek.

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
df <- read.csv("data/unpopular_songs.csv")
summary(df)
```

```
##   danceability        energy               key            loudness      
##  Min.   :0.0000   Min.   :0.0000203   Min.   : 0.000   Min.   :-51.808  
##  1st Qu.:0.4420   1st Qu.:0.3790000   1st Qu.: 2.000   1st Qu.:-13.796  
##  Median :0.6020   Median :0.5690000   Median : 5.000   Median : -9.450  
##  Mean   :0.5725   Mean   :0.5497713   Mean   : 5.223   Mean   :-11.359  
##  3rd Qu.:0.7300   3rd Qu.:0.7450000   3rd Qu.: 9.000   3rd Qu.: -6.726  
##  Max.   :0.9860   Max.   :1.0000000   Max.   :11.000   Max.   :  3.108  
##       mode        speechiness      acousticness    instrumentalness  
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000000  
##  1st Qu.:0.000   1st Qu.:0.0384   1st Qu.:0.0365   1st Qu.:0.000000  
##  Median :1.000   Median :0.0589   Median :0.2330   Median :0.000133  
##  Mean   :0.641   Mean   :0.1380   Mean   :0.3542   Mean   :0.232943  
##  3rd Qu.:1.000   3rd Qu.:0.1880   3rd Qu.:0.6570   3rd Qu.:0.517000  
##  Max.   :1.000   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      
##  Length:10877       Min.   : 0.000   Length:10877       Length:10877      
##  Class :character   1st Qu.: 1.000   Class :character   Class :character  
##  Mode  :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 we largely have quantitative data, with a few exceptions. Not all of these are useful, but we'll make whether or not its explicit a factor for now, as well as popularity (after we look at correlation). We'll also look for correlated values.

```
df$explicit <- as.factor(df$explicit)
summary(df)
```

```
##   danceability        energy               key            loudness
##  Min.   :0.0000   Min.   :0.0000203   Min.   : 0.000   Min.   :-51.808
##  1st Qu.:0.4420   1st Qu.:0.3790000   1st Qu.: 2.000   1st Qu.:-13.796
##  Median :0.6020   Median :0.5690000   Median : 5.000   Median : -9.450
##  Mean   :0.5725   Mean   :0.5497713   Mean   : 5.223   Mean   :-11.359
##  3rd Qu.:0.7300   3rd Qu.:0.7450000   3rd Qu.: 9.000   3rd Qu.: -6.726
##  Max.   :0.9860   Max.   :1.0000000   Max.   :11.000   Max.   :  3.108
##       mode        speechiness      acousticness    instrumentalness
##  Min.   :0.000   Min.   :0.0000   Min.   :0.0000   Min.   :0.000000
##  1st Qu.:0.000   1st Qu.:0.0384   1st Qu.:0.0365   1st Qu.:0.000000
##  Median :1.000   Median :0.0589   Median :0.2330   Median :0.000133
##  Mean   :0.641   Mean   :0.1380   Mean   :0.3542   Mean   :0.232943
##  3rd Qu.:1.000   3rd Qu.:0.1880   3rd Qu.:0.6570   3rd Qu.:0.517000
##  Max.   :1.000   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
##
##
##
```

```
cor(df[c(1,2,3,4,5,6,7,8,9,10,11,12,14)])
```

```
##                   danceability      energy          key     loudness
## danceability      1.0000000000  0.10357554  0.001416440  0.384798006
## energy            0.1035755370  1.00000000  0.032847557  0.668247944
## key               0.0014164396  0.03284756  1.000000000  0.020238291
## loudness          0.3847980060  0.66824794  0.020238291  1.000000000
## mode             -0.0424166570 -0.04371262 -0.174170158  0.007144594
## speechiness       0.2880560637  0.06065882 -0.003339108  0.067091927
## acousticness     -0.2537596673 -0.57807060 -0.017360855 -0.491999477
## instrumentalness -0.3345776576 -0.31475687 -0.026367389 -0.547322987
## liveness         -0.2502105046  0.25837921 -0.001745424 -0.018978820
## valence           0.5171426279  0.31726610  0.015964344  0.426772633
## tempo             0.0900580502  0.17122835 -0.003040262  0.202227504
## duration_ms       0.0004830046  0.15201424  0.006044278  0.195281479
## popularity        0.1597255536  0.05469420 -0.002388392  0.149949613
##                           mode  speechiness acousticness instrumentalness
## danceability     -0.0424166570  0.288056064  -0.25375967     -0.334577658
## energy           -0.0437126214  0.060658817  -0.57807060     -0.314756871
## key              -0.1741701578 -0.003339108  -0.01736086     -0.026367389
## loudness          0.0071445943  0.067091927  -0.49199948     -0.547322987
## mode              1.0000000000 -0.087636772   0.03888040     -0.063920945
## speechiness      -0.0876367717  1.000000000  -0.11592434     -0.273849185
## acousticness      0.0388803990 -0.115924341   1.00000000      0.291033539
## instrumentalness -0.0639209452 -0.273849185   0.29103354      1.000000000
## liveness         -0.0241449112  0.050249663  -0.02456814     -0.008284127
## valence           0.0002389504  0.115257854  -0.21538759     -0.335547352
## tempo             0.0171224145  0.038543375  -0.18312285     -0.119385544
## duration_ms       0.0351389868 -0.098355503  -0.11730165     -0.148671815
## popularity       -0.0454684641  0.050489909  -0.11698471     -0.075279942
##                      liveness       valence        tempo   duration_ms
## danceability     -0.250210505  0.5171426279  0.090058050  0.0004830046
## energy            0.258379213  0.3172660977  0.171228345  0.1520142437
## key              -0.001745424  0.0159643436 -0.003040262  0.0060442781
## loudness         -0.018978820  0.4267726333  0.202227504  0.1952814794
## mode             -0.024144911  0.0002389504  0.017122414  0.0351389868
## speechiness       0.050249663  0.1152578541  0.038543375 -0.0983555028
## acousticness     -0.024568144 -0.2153875874 -0.183122846 -0.1173016518
## instrumentalness -0.008284127 -0.3355473521 -0.119385544 -0.1486718149
## liveness          1.000000000 -0.1129996078 -0.029490757  0.0683864612
## valence          -0.112999608  1.0000000000  0.172984416  0.0460316403
## tempo            -0.029490757  0.1729844162  1.000000000  0.0509919444
## duration_ms       0.068386461  0.0460316403  0.050991944  1.0000000000
## popularity       -0.066955096  0.0358241022  0.061602311 -0.0250484441
##                    popularity
## danceability      0.159725554
## energy            0.054694203
## key              -0.002388392
## loudness          0.149949613
## mode             -0.045468464
## speechiness       0.050489909
## acousticness     -0.116984708
## instrumentalness -0.075279942
## liveness         -0.066955096
## valence           0.035824102
## tempo             0.061602311
## duration_ms      -0.025048444
## popularity        1.000000000
```

```
df$popularity <- as.factor(df$popularity)
```

We don't see a ton of clearly related values, though how many attributes we have does make it difficult to read. We'll hope that the algorithms will do well at reducing the amount of attributes we have entering into this data.

Let's take a closer look at popularity, now that its factored.

```
summary(df$popularity)
```

```
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2694 2101 2146 1494  457  309  212  137  112   59   80   45   59  248  544  152
##   16   17   18
##   19    5    4
```

Hmm, a few too many factors. Let's combine some of these with respect to how many are in each category.

```
#install.packages("forcats")
library(forcats)
popularityclass <- fct_collapse(df$popularity, horrible=c('0','1'), bad=c('2','3','4','5'),
       meh=c('6','7','8','9','10','11','12'), passable=c('13','14','15','16','17','18'))

df$popclass <- popularityclass
```

And now we’ll be sure it worked.

```
summary(df$popclass)
```

```
## horrible      bad      meh passable 
##     4795     4406      704      972
```

```
names(df)
```

```
##  [1] "danceability"     "energy"           "key"              "loudness"        
##  [5] "mode"             "speechiness"      "acousticness"     "instrumentalness"
##  [9] "liveness"         "valence"          "tempo"            "duration_ms"     
## [13] "explicit"         "popularity"       "track_name"       "track_artist"    
## [17] "track_id"         "popclass"
```

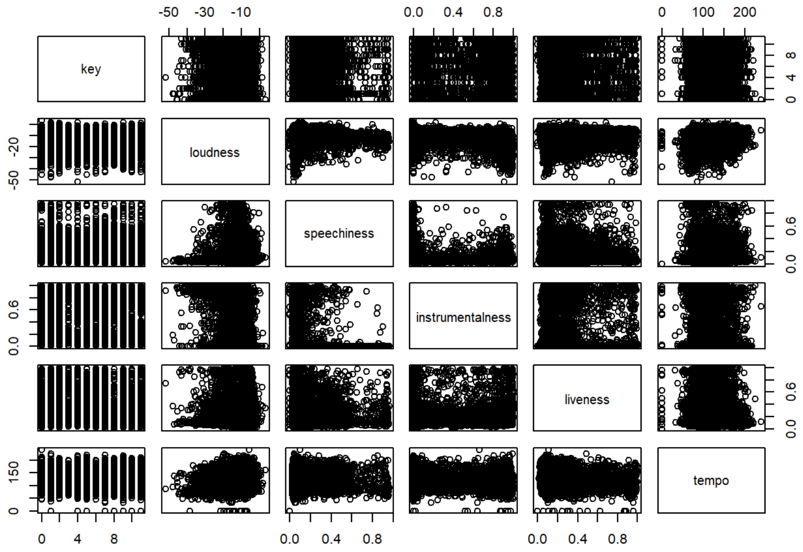
Cheers! Let’s separate it into training data now.

```
i <- sample(1:nrow(df),nrow(df)*.8,replace=FALSE)
train <- df[i,]
test <- df[-i,]
```

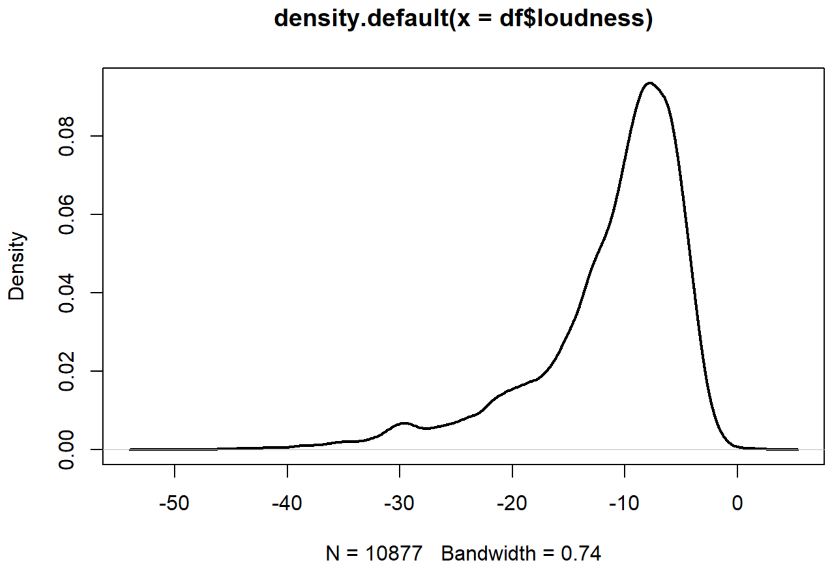
VISUAL EXPLORATION

Now, let’s look at some charts to understand things a bit better.

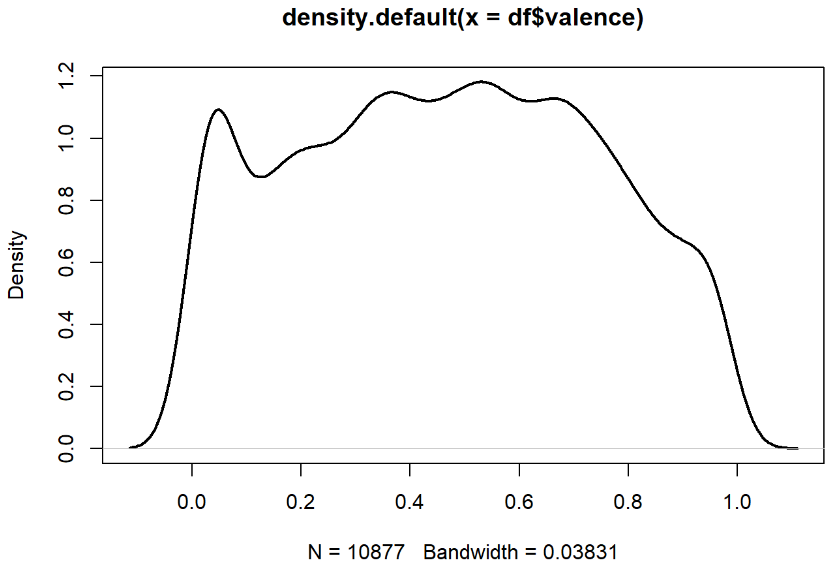
```
pairs(df[c(3,4,6,8,9,11)])
```



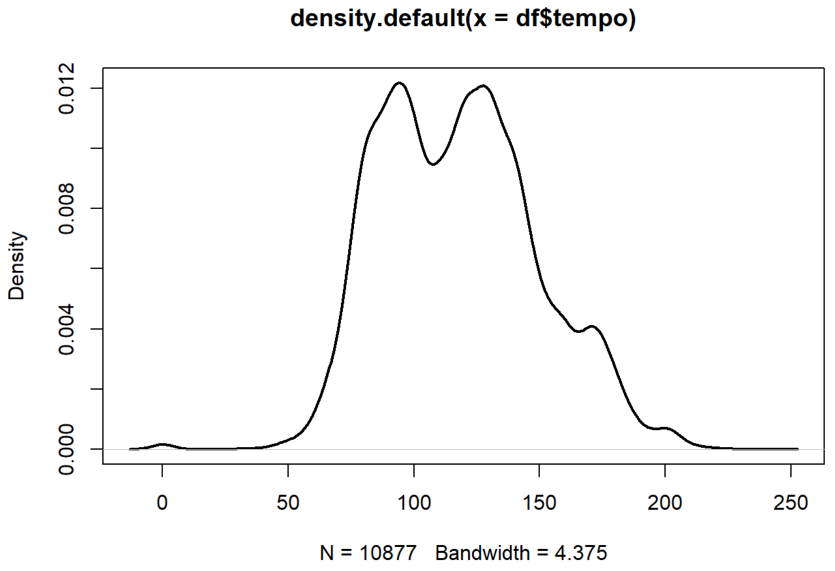
```
plot(density(df$loudness),lwd=2)
```



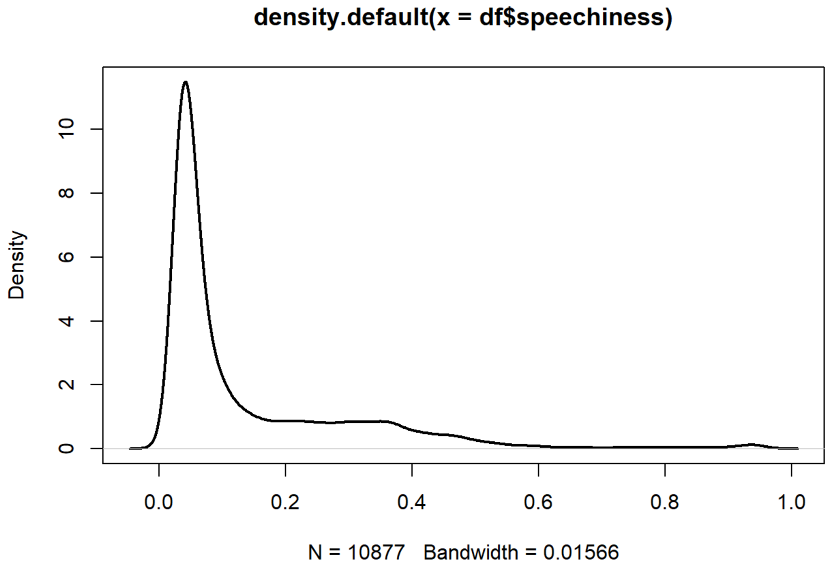
```
plot(density(df$valence),lwd=2)
```



```
plot(density(df$tempo),lwd=2)
```



```
plot(density(df$speechiness),lwd=2)
```



We confirm that key, liveliness, and tempo are not very useful. We can now better understand how the data is laid out, and confirmed that correlation is difficult to find. This is why we will be using a kNN model to test dimensionality on this data.

DIMENSIONALITY ALGORITHMS

Okay, now let's run PCA on the data. We have a lot of columns to consider. We'll center and scale them while we're at it.

```
set.seed(2022)
pca_out <- preProcess(train[,1:10], method=c("center","scale","pca"),k=5)
pca_out
```

```
## Created from 8701 samples and 10 variables
## 
## Pre-processing:
##   - centered (10)
##   - ignored (0)
##   - principal component signal extraction (10)
##   - scaled (10)
## 
## PCA needed 9 components to capture 95 percent of the variance
```

We weren't able to remove much.

Let's plot what we got. We'll put them on 3 separate 3d charts.

```
train_pc <- predict(pca_out,train[,1:10])
test_pc <-  predict(pca_out, test[,1:10])

#install.packages("plotly")
library(plotly)
```

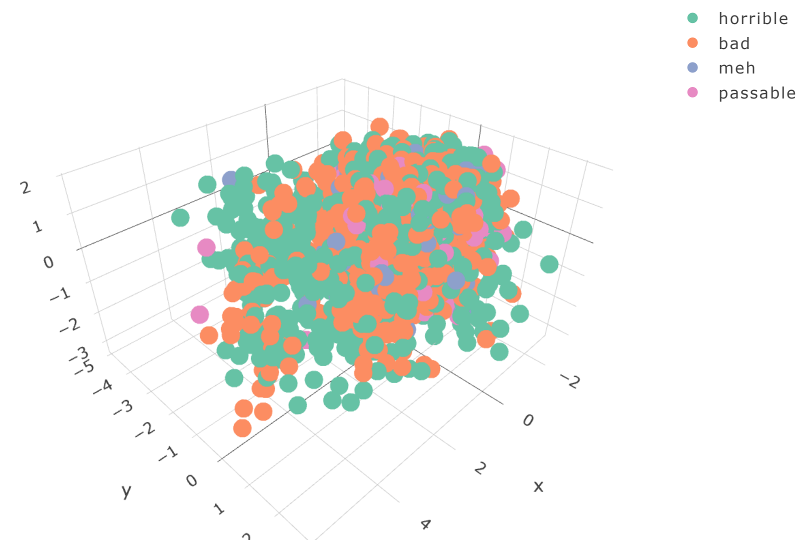
```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

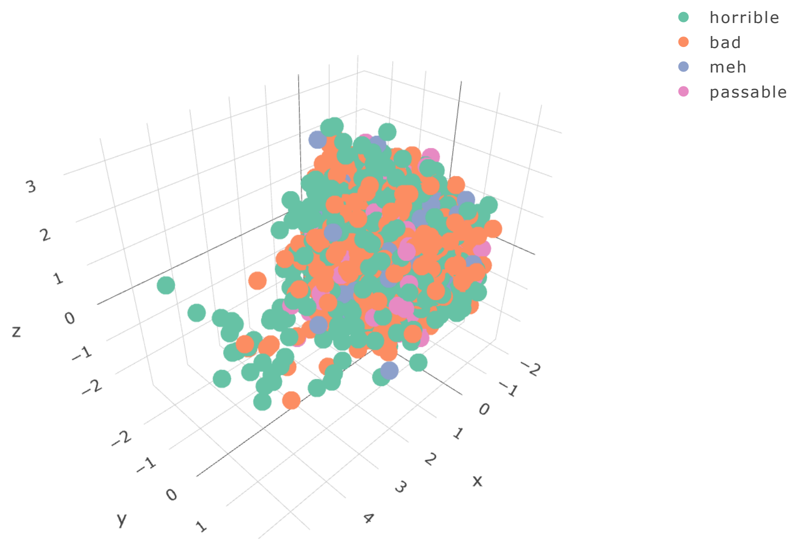
```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

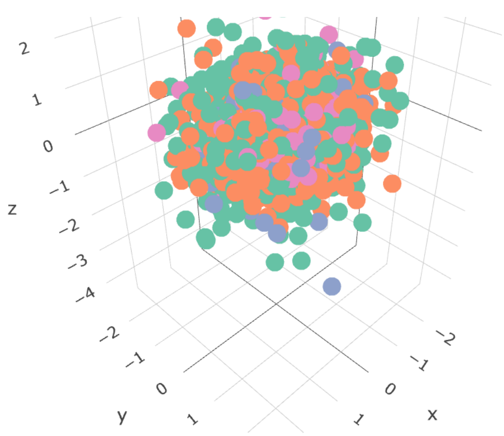
```
plot_ly(x=test_pc$PC1, y=test_pc$PC2, z=test_pc$PC3, type="scatter3d", mode="markers",color=test$popclass)
```



```
plot_ly(x=test_pc$PC4, y=test_pc$PC5, z=test_pc$PC6, type="scatter3d", mode="markers",color=test$popclass)
```



```
plot_ly(x=test_pc$PC7, y=test_pc$PC8, z=test_pc$PC9, type="scatter3d", mode="markers",color=test$popclass)
```



Things are not looking promising. We can hope that since it wasn't able to reduce much, though, that using all the predictors it created will help more, even if we can't visualize it.

Let's try kNN on it.

```
library(class)
train_df <-
        data.frame(train_pc$PC1,train_pc$PC2,train_pc$PC3,train_pc$PC4,train_pc$PC5,train_pc$PC6,train_pc$PC7,train_pc$PC8,t
        train$popclass)

test_df <-
        data.frame(test_pc$PC1,test_pc$PC2,test_pc$PC3,test_pc$PC4,test_pc$PC5,test_pc$PC6,test_pc$PC7,test_pc$PC8,t
        est_pc$PC9, test$popclass)

predknn <- knn(train=train_df[,1:9], test=test_df[,1:9], cl=train_df[,10], k=5)
mean(predknn==test$popclass)
```

```
## [1] 0.4852941
```

```
confusionMatrix(data=predknn, reference=test$popclass)
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction horrible bad meh passable
##   horrible       544 368  54       76
##   bad            348 498  62       89
##   meh             18  14   5        6
##   passable        33  44   8        9
##
## Overall Statistics
##
##                   Accuracy : 0.4853
##                     95% CI : (0.4641, 0.5065)
##        No Information Rate : 0.4334
##        P-Value [Acc > NIR] : 6.192e-07
##
##                      Kappa : 0.1323
##
##   Mcnemar's Test P-Value : 1.880e-15
##
## Statistics by Class:
##
##                      Class: horrible Class: bad Class: meh Class: passable
## Sensitivity                   0.5769     0.5390   0.038760        0.050000
## Specificity                   0.5961     0.6014   0.981436        0.957415
## Pos Pred Value                0.5221     0.4995   0.116279        0.095745
## Neg Pred Value                0.6481     0.6387   0.941866        0.917867
## Prevalence                    0.4334     0.4246   0.059283        0.082721
## Detection Rate                0.2500     0.2289   0.002298        0.004136
## Detection Prevalence          0.4789     0.4582   0.019761        0.043199
## Balanced Accuracy             0.5865     0.5702   0.510098        0.503707
```

Well, this doesn't seem like it was too helpful. We have a less than 50% chance of getting our classification correct, even we're looking at our larger trained classes. This well may be simply due to poor correlation in data, however. We weren't even able to reduce the data much. On another data set, PCA may be more beneficial.

LINEAR DISCRIMINANT ANALYSIS

Let's see if LDA works better for our data set. However, we know well that out data is not linear, so hopes are low.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:plotly':
##
##     select
```

```
ldapop <- MASS::lda(x=train[,1:12],grouping=train$popclass, data=train)
#ldapop <- lda(train$popclass~., data=train)
ldapop$means
```

```
##          danceability    energy      key   loudness      mode speechiness
## horrible    0.5440875 0.5471038 5.318536 -12.428731 0.6542056   0.1432161
## bad         0.5804911 0.5367301 5.237507 -10.940616 0.6496267   0.1228314
## meh         0.6071285 0.6039334 4.968696  -9.113452 0.6626087   0.1495861
## passable    0.6549184 0.5773484 5.305556  -9.391605 0.5429293   0.1741463
##          acousticness instrumentalness  liveness   valence    tempo duration_ms
## horrible    0.3897951        0.2513320 0.2324003 0.4443876 116.4034    202060.6
## bad         0.3471737        0.2352933 0.1956088 0.4788491 118.2083    212001.6
## meh         0.2722616        0.1311003 0.2015706 0.4814607 122.3585    218305.2
## passable    0.2690076        0.1862304 0.1900961 0.4776485 120.6720    181039.0
```

Means were found well, and everything looks good. We have to break it up for the sake of Plotly syntax, as it seemed to have some confusion due to commas in predictor names. PCA was strictly dimension reduction, but LDA also predicts, so we won't be using kNN this time.

```
lda_pred <- predict(ldapop,newdata=test[,1:12],type="class")
head(lda_pred$class)
```

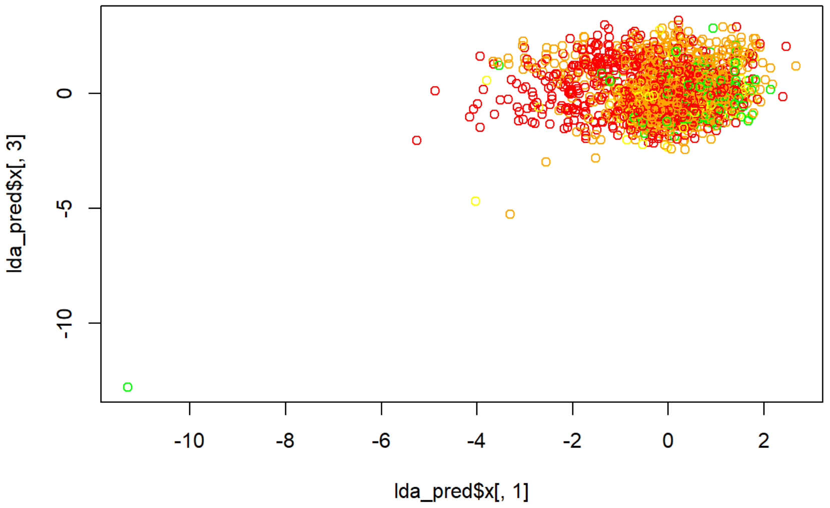
```
## [1] horrible bad      bad      horrible bad      bad
## Levels: horrible bad meh passable
```

```
#lda_train <- predict(ldapop,data=train,type="class")
```

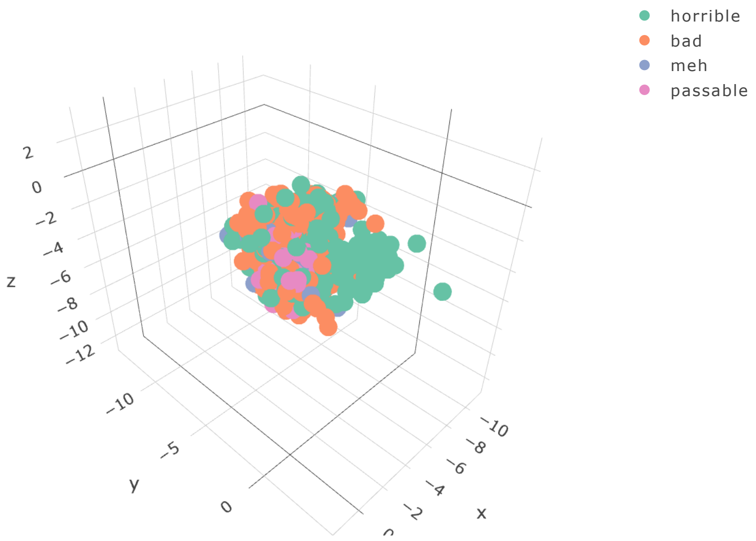
We know the majority of our data is in the 'bad' or 'horrible' range, so all looks good here.

Now, let's plot it!

```
library(plotly)
plot(lda_pred$x[,1], lda_pred$x[,3], pch=c(16,17,18,15)[unclass(test_pc$popclass)],
       col=c("red","orange","yellow","green")[unclass(test$popclass)])
```



```
xaxis <- lda_pred$x[,1]
yaxis <- lda_pred$x[,2]
zaxis <- lda_pred$x[,3]
target<- test$popclass
plot_ly(x=xaxis,y=yaxis,z=zaxis,type="scatter3d",mode="markers",color=target)
```



Things are not looking promising. It looks largely the same as any of our charts from principal components, even though we were able to chart all the attributes that were produced to see a visible appearance in one go this time.

We now can check our confusion matrix and look into how well we actually managed to predict data.

```
library(class)
mean(lda_pred$class==test$popclass)
```

```
## [1] 0.4779412
```

```
confusionMatrix(data=lda_pred$class, reference=test$popclass)
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction horrible bad meh passable
##   horrible      551 435  68       83
##   bad           392 489  61       97
##   meh             0   0   0        0
##   passable        0   0   0        0
## 
## Overall Statistics
## 
##                Accuracy : 0.4779          
##                  95% CI : (0.4568, 0.4992)
##     No Information Rate : 0.4334          
##     P-Value [Acc > NIR] : 1.58e-05        
## 
##                   Kappa : 0.0854          
## 
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: horrible Class: bad Class: meh Class: passable
## Sensitivity                   0.5843     0.5292    0.00000         0.00000
## Specificity                   0.5247     0.5607    1.00000         1.00000
## Pos Pred Value                0.4846     0.4706        NaN             NaN
## Neg Pred Value                0.6227     0.6174    0.94072         0.91728
## Prevalence                    0.4334     0.4246    0.05928         0.08272
## Detection Rate                0.2532     0.2247    0.00000         0.00000
## Detection Prevalence          0.5225     0.4775    0.00000         0.00000
## Balanced Accuracy             0.5545     0.5450    0.50000         0.50000
```

The model entirely failed for ‘okay’ and ‘passable’ songs, which is not surprising considering our model visualization. It did slightly better than PCA with kNN, however. We are effectively worse than a coin flip, made worse only by there being 4 potential classes to choose from.

CONCLUSION AND ANALYSIS

We chose this data since it being advertised for clustering made it seem like it would be good for kNN as well, and that the reduction would help simplify the large number of attributes. However, after interacting with it, this expectation was folly on our part. There is more that goes into making a dataset good for kNN. Thinking about the nature of our data, of bad songs on Spotify, we can also conclude that there isn’t a ton of trend with what makes a song “bad”. Perhaps from this data a genre may be able to be found via clustering, but popularity isn’t an equation of things such as tempo, energy, instruments, or anything else. Sometimes a song is just bad for content or other reasons. When it came down to it, PCA+kNN and LDA effectively made a coin flip then rated a song as ‘bad’ or ‘horrible’. While the PCA attempt was able to occasionally succeed for the smaller classes, LDA may well have been more accurate due to the fact that it stuck to the larger classes and did not try to sort anything into the smaller classes. Since the values were so scattered, increasing the amount of data likely would not have helped significantly. The reality of it is that there is not much correlation, and that we have learned that PCA nor LDA is able to find or create correlation where there is none.

---

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/) ↩