mini_3

Raunaq Rewari

December 7, 2017

R Markdown

Classification

We want to predict whether a song will ever make it to the top 20. Since a song's popularity should intuitively depend on the inherent features of a song, we used Spotify's API to get the following features per song: Danceability Energy Key Loudness Mode Speechiness Acousticness Instrumentalness Liveness Valence Tempo

For a first attempt, we focused on one region, say US, and all songs that were ever in the top 20 were given a label 1 and the others a label 0. We used logistic regression to model the problem with the covariates mentioned above. We found no correlation between popularity and any of the covariates.

We next looked at the top 6 popular Regions, again trying to predict if a song will make it to the top 20. We added the Region covariate this time into our analysis and again found no statistically significant covariate. Following is a plot that shows (the lack of) correlation between each covariate and the mean number of streams for each song.

```r
dt = read.csv("data.csv")
song_feats = read_csv("song_feats.csv")
```

```
## Parsed with column specification:
## cols(
##   URL = col_character(),
##   energy = col_double(),
##   liveness = col_double(),
##   tempo = col_double(),
##   speechiness = col_double(),
##   acousticness = col_double(),
##   instrumentalness = col_double(),
##   time_signature = col_integer(),
##   danceability = col_double(),
##   key = col_integer(),
##   duration_ms = col_integer(),
##   loudness = col_double(),
##   valence = col_double()
## )
```

```r
# Number of unique songs in the top x. Find the region with the maximum number
x = 20
unique_top_20 = dt %>% group_by(Region) %>% filter(Position %in% c(1:x)) %>%
  summarise(Total_Unique = n_distinct(Track.Name)) %>% arrange(desc(Total_Unique))
```

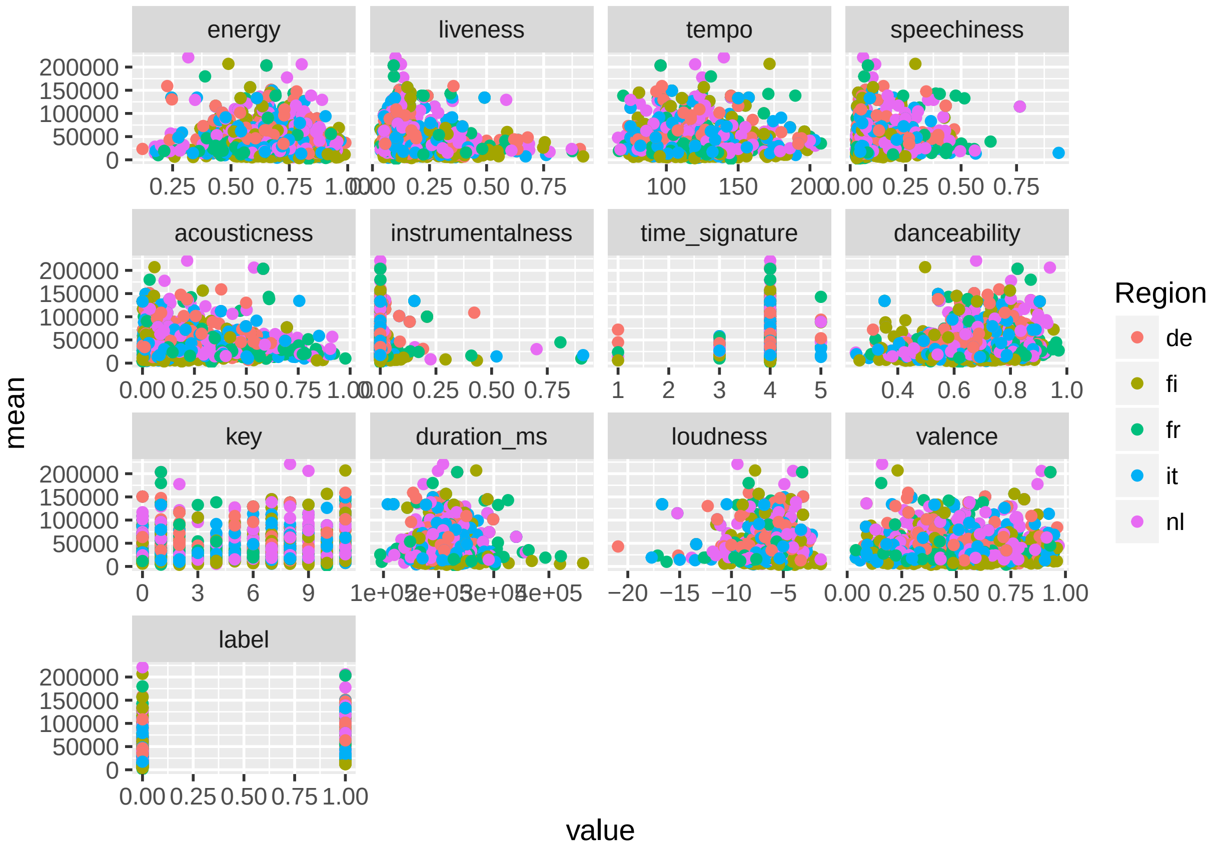
```
## Warning: package 'bindrcpp' was built under R version 3.2.5
```

```r
# Get names of tracks in the region decided from above that are in the top at least once
top_track_names = unique((dt %>% filter((Region == "fr") & (Position %in% c(1:x))))$URL)

# Get bottom tracks
bottom_track_names = unique((dt %>% filter((Region == "fr") & !(URL %in% top_track_names)))$URL)
bottom_track_names = sample(bottom_track_names, length(top_track_names))
```

```
# Make new dataset with labels added
new_dt = song_feats %>% filter((URL %in% top_track_names) | (URL %in% bottom_track_names))
new_dt = transform(new_dt, label = if_else(URL %in% top_track_names, 1, 0))
new_col = rep("fr", nrow(new_dt))
new_dt$Region = new_col

regions_to_include = c("fi", "nl", "it", "de")
for (i in (1 : length(regions_to_include))){
  top_track_names = unique((dt %>% filter((Region == regions_to_include[i]) & (Position %in% c(1:x))))$U
  bottom_track_names = unique((dt %>% filter((Region == regions_to_include[i]) & !(URL %in% top_track_na
  bottom_track_names = sample(bottom_track_names, length(top_track_names))
  curr_dt = song_feats %>% filter((URL %in% top_track_names) | (URL %in% bottom_track_names))
  print(nrow(curr_dt))
  curr_dt = transform(curr_dt, label = if_else(URL %in% top_track_names, 1, 0))
  new_col = rep(regions_to_include[i], nrow(curr_dt))
  curr_dt$Region = new_col
  new_dt = rbind(new_dt, curr_dt)
  print(nrow(new_dt))
}
```

```
## [1] 264
## [1] 527
## [1] 238
## [1] 765
## [1] 224
## [1] 989
## [1] 199
## [1] 1188
```

```
mean_streams = dt %>% group_by(URL) %>% summarise(mean = mean(Streams))
songSummaries5Regions = merge(new_dt,mean_streams,by="URL")

pivoted = melt(songSummaries5Regions,id.vars = c("mean","URL","Region"),
               value.name = "value")

ggplot(pivoted)+geom_point(aes(x=value,y=mean,color=Region))+
  facet_wrap(~variable,scales = "free_x")
```



```
akash_data = read.csv("data_t20regions.csv")

confusion_matrix = function(fitted.results, new_dt){
  num_false_positive = 0
  num_false_negative = 0
  num_true_negative = 0
  num_true_positive = 0
  for (i in 1:length(fitted.results)){
    if ((fitted.results[i] == 1) & (new_dt$label[i]) == 0){
      num_false_positive = num_false_positive + 1
    }
    if ((fitted.results[i] == 0) & (new_dt$label[i]) == 1){
      num_false_negative = num_false_negative + 1
    }
    if ((fitted.results[i] == 0) & (new_dt$label[i]) == 0){
      num_true_negative = num_true_negative + 1
    }
    if ((fitted.results[i] == 1) & (new_dt$label[i]) == 1){
      num_true_positive = num_true_positive + 1
    }
  }
  print(paste('False Positive Rate: ',num_false_positive/length(fitted.results)))
  print(paste('False Negative Rate: ',num_false_negative/length(fitted.results)))
  print(paste('True Negative Rate: ',num_true_negative/length(fitted.results)))
  print(paste('True Positive Rate: ',num_true_positive/length(fitted.results)))
}
```

```r
x = 10
num_regions = 2
unique_songs = akash_data %>% group_by(URL) %>% filter(Position %in% c(1:x)) %>% summarise(num_unique =

top_track_URLS = (unique_songs %>% filter(num_unique >= num_regions))$URL

#bottom_track_URLS = (unique_songs %>% filter(num_unique == 1))$URL
bottom_track_URLS = (song_feats %>% filter(!(URL %in% unique_songs$URL)))$URL
#bottom_track_URLS = sample(bottom_track_URLS, length(top_track_URLS))

percent_test = 0.2
top_test_URLS = sample(top_track_URLS, percent_test*length(top_track_URLS))
top_train_URLS = top_track_URLS[!top_track_URLS %in% top_test_URLS]

bottom_test_URLS = sample(bottom_track_URLS, length(top_test_URLS))
bottom_train_URLS = bottom_track_URLS[!bottom_track_URLS %in% bottom_test_URLS]
bottom_train_URLS = sample(bottom_train_URLS, length(top_train_URLS))

# Create train and test set
train_data = song_feats %>% filter((URL %in% top_train_URLS) | (URL %in% bottom_train_URLS))
train_data = transform(train_data, label = if_else(URL %in% top_train_URLS, 1, 0))

test_data = song_feats %>% filter((URL %in% top_test_URLS) | (URL %in% bottom_test_URLS))
test_data = transform(test_data, label = if_else(URL %in% top_test_URLS, 1, 0))

to_predict_train = train_data[,c("energy","liveness","tempo","speechiness","acousticness","instrumental
to_predict_test = test_data[,c("energy","liveness","tempo","speechiness","acousticness","instrumentalne
```

We came to the conclusion that we needed a better decision boundary between what we label as popular and what we label as not. In other words, the features we had to work with did not have enough signal. Finally we decided to label songs as popular if they were in the top 20 in at least 5 of the top 20 highest streaming regions. Songs that did not make it to the highest streaming regions composed the negative set for our classification problem.

Predicting on a test set

```r
classification = glm(label ~ energy + liveness + tempo + speechiness + acousticness + instrumentalness +

fitted.results.train <- predict(classification, newdata=to_predict_train, type='response')
fitted.results.train <- ifelse(fitted.results.train > 0.5,1,0)

misClasificError.train <- mean(fitted.results.train != train_data$label)
print(paste('Training Accuracy',1-misClasificError.train))
```

```
## [1] "Training Accuracy 0.699570815450644"
```

```r
fitted.results.test <- predict(classification, newdata=to_predict_test, type='response')
fitted.results.test <- ifelse(fitted.results.test > 0.5,1,0)

misClasificError.test <- mean(fitted.results.test != test_data$label)
print(paste('Testing Accuracy',1-misClasificError.test))
```

```
## [1] "Testing Accuracy 0.637931034482759"
```

Inference. Part a): Model summary

```
summary(classification)
```

```
##
## Call:
## glm(formula = label ~ energy + liveness + tempo + speechiness +
##     acousticness + instrumentalness + danceability + loudness +
##     valence, family = binomial, data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0744  -0.9532  -0.5259   1.0133   2.8291
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       4.554599   2.013847   2.262  0.02372 *
## energy           -4.658370   1.561766  -2.983  0.00286 **
## liveness         -1.610557   1.251155  -1.287  0.19801
## tempo            -0.004578   0.005219  -0.877  0.38045
## speechiness       1.374758   1.472933   0.933  0.35064
## acousticness     -2.751759   0.851212  -3.233  0.00123 **
## instrumentalness -3.534038   3.587158  -0.985  0.32453
## danceability      3.321900   1.303506   2.548  0.01082 *
## loudness          0.359289   0.109677   3.276  0.00105 **
## valence          -0.907602   0.829851  -1.094  0.27409
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 318.87  on 232  degrees of freedom
## Residual deviance: 271.39  on 223  degrees of freedom
## AIC: 291.39
##
## Number of Fisher Scoring iterations: 5
```

Inference. Part b): Fitting model on test data

```
classification_test = glm(label ~ energy + liveness + tempo + speechiness + acousticness + instrumental
summary(classification_test)
```

```
##
## Call:
## glm(formula = label ~ energy + liveness + tempo + speechiness +
##     acousticness + instrumentalness + danceability + loudness +
##     valence, family = binomial, data = test_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9887  -0.9228  -0.3601   1.0443   1.7809
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -0.30504    4.87512  -0.063   0.9501
## energy           -0.42470    3.58120  -0.119   0.9056
## liveness         -1.19326    3.10806  -0.384   0.7010
## tempo            -0.01186    0.01255  -0.945   0.3446
## speechiness       1.70686    3.38646   0.504   0.6142
## acousticness     -2.26950    1.99714  -1.136   0.2558
## instrumentalness -0.29143    2.91980  -0.100   0.9205
## danceability      5.96670    3.48983   1.710   0.0873 .
## loudness          0.18403    0.24482   0.752   0.4522
## valence          -1.08588    1.98955  -0.546   0.5852
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 79.783  on 57  degrees of freedom
## Residual deviance: 67.028  on 48  degrees of freedom
## AIC: 87.028
##
## Number of Fisher Scoring iterations: 5
```

Inference. Part c): Bootstrapping coefficients of regression for 1000 samples

```
num_samples = 1000

intercept = c()
energy = c()
liveness = c()
tempo = c()
speechiness = c()
acousticness = c()
instrumentalness = c()
danceability = c()
loudness = c()
valence = c()

for (i in 1:num_samples){
top_train_URLS_boot = sample(top_train_URLS, length(top_train_URLS), replace = TRUE)
bottom_train_URLS_boot = sample(bottom_train_URLS, length(bottom_train_URLS), replace = TRUE)

# Create train and test set
train_data_boot = song_feats %>% filter((URL %in% top_train_URLS_boot) | (URL %in% bottom_train_URLS_bo
train_data_boot = transform(train_data_boot, label = if_else(URL %in% top_train_URLS_boot, 1, 0))

classification = glm(label ~ energy + liveness + tempo + speechiness + acousticness + instrumentalness +
intercept = c(intercept, classification$coefficients[1])
energy = c(energy, classification$coefficients[2])
liveness = c(liveness, classification$coefficients[3])
tempo = c(tempo, classification$coefficients[4])
speechiness = c(speechiness, classification$coefficients[5])
acousticness = c(acousticness, classification$coefficients[6])
instrumentalness = c(instrumentalness, classification$coefficients[7])
danceability = c(danceability, classification$coefficients[8])
loudness = c(loudness, classification$coefficients[9])
valence = c(valence, classification$coefficients[10])
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
# Specify whichever covariate you need to find the confidence intervals for
quantile(loudness, 0.025)
```

```
##      2.5%
## 0.2015614
```

```
quantile(loudness, 0.975)
```

```
##     97.5%
## 0.5837501
```