Music Lyrics Genre Classifier

Submitted to:

Michael LuValle, PhD

Statistics 486

May 5, 2023

By:

Anish Gupta

Ari Morrison

Jessica Chen

Luke Beebe

Tyra Lassiter

Music Lyrics Genre Classifier

Motivations

Throughout our Applied Statistical Learning course, we have covered many different methods for statistical analysis. As a group, we came to the conclusion that the statistical method that piqued our interest was Natural Language Processing (NLP). We have seen NLP in action through chatbots and email filtering functions, and now was the opportunity for us to create our own NLP classification project to use as a template for future projects and explore a topic that most people enjoy, music. After determining that NLP was our statistical method of choice, we also determined that music lyrics would be our project topic and we were curious to see if there was a relationship between music lyrics and decades, or music lyrics and genre.

Preliminary Analysis

We began preliminary analysis by evaluating our dataset. We cross referenced our initial dataset with internet lyric sources and determined that it was not reliable. Eventually, we found a reliable dataset and used Random Forest (RF) and K Nearest Neighborhood (KNN) for our analysis.

We chose RF because of its reputation as being the simplest, quickest algorithms to work with. Because it is a classification system based on bootstrapped samples, it gives you a sense of what the data looks like on average, although it might not be the strongest at predicting. It is a great starting point, and most of the other examples we researched also followed this thought.

We also used KNN because intuitively it made sense for this project. It plots multidimensional data and plots each input among the other data points. It then finds the 'nearest neighbor' and predicts it because of its similarity.

These both gave insights into the data. While predicting decades based on lyrics we encountered initial success, but while looking at our results (38% for KNN, 51% for RF), we saw it mostly predicted 2010, whose songs represented half the sample we were studying. This was a problem.

To fix this problem we created a function that took a skewed set and separated and weighted each category until the total word distribution was close to uniform. After testing this data we had impressive results (89% on RF). However, we were cheating. What we did by making the data 'uniform' was amplify the word count for each song. So, a song from the 1960s said 'hello' 21 times instead of 3 times. Although it made the decade word count uniform, it made the songs from the 1960s word count obscenely high and easier for the models to predict because the test data was now skewed the same way. We had to think of another way to make the data uniform between the categories we were comparing.

We then resorted to taking a uniform sample from the original dataset. This was the most straightforward process to circumnavigate the aforementioned problems.

Decade has 7 categories and genre has 6. So a random guess has a 14% chance of predicting the decade of a song and about a 17% chance of guessing the genre correctly.

```
table(song.data$decade)
1960  1970  1980  1990  2000  2010  2020
10000 10000 10000 10000 10000 10000 10000

table(song.data1$genre)
country    misc     pop     rap      rb    rock
  10000   10000   10000   10000   10000   10000
```

After taking a uniform sample, we decided to run tests on the data. We somewhat arbitrarily chose to set *removeSparseTerms* to 0.99. This function removes words that occur very infrequently in the data. In this case, it removes all the least common words until 1% of all the words have been removed. We experimented with this parameter later on.

After that, we tested RF and KNN on both the decades and the genres. RF with 10 trees, gave us 26% accuracy on the decades and 48% accuracy with genre. KNN, with $k = 13$ and *removeSparseTerms* = .95, gave us 21% accuracy with decade and 38% accuracy with genre.

Seeing our relative success at predicting genre, we decided to dedicate the rest of our project to fine tuning the parameters to better predict genre. Reflecting, it makes a lot of intuitive sense that lyrics are more correlated to genre than decade.

The next step was testing different RF and KNN parameters and determine which would give us the most accurate model. The search for the best parameters was a manual, trial and error task that used data entry in Excel. When predicting genre, we determined that the optimal parameter for model accuracy was *ntrees* = 200 for RF, with *removeSparseTerms* = 0.99. The optimal parameters for KNN was $k = 13$, with *removeSparseTerms* = 0.95. Below are the test runs and results that lead us to these conclusions.

RUNS	model	RemoveSparse	ntree	Results
1	RF	0.95	10	0.46
2	RF	0.975	10	0.476
3	RF	0.925	10	0.459
4	RF	0.9	10	0.445
5	RF	0.8	10	0.4
6	RF	0.99	10	0.4847
7	RF	0.999	10	0.4846
8	RF	0.99	5	0.436
9	RF	0.99	15	0.5046
10	RF	0.99	20	0.519
11	RF	0.99	50	0.5475
12	RF	0.99	100	0.56
13	RF	0.99	200	0.5606

	model	RemoveSparse	k	Results
1	Knn	0.95	13	0.379
2	Knn	0.975	8	0.363
3	Knn	0.925	5	0.372
4	Knn	0.9	5	0.37
5	Knn	0.8	5	0.358
6	Knn	0.99	10	0.3545
7	Knn	0.95	20	0.355
8	Knn	0.95	5	0.3447
9	Knn	0.95	10	0.3556

Process

Data Sorting & Cleaning

The best dataset we found was a Genius Dataset with 3 million rows on Kaggle. This dataset was 3GB and not feasible to work with on a laptop. One of our group members randomly sampled 70,000 rows of the data on a built PC. They then distributed the sampled dataset to the team. It had many columns, but the columns we needed were the ID, Title, Genre, Lyrics, and Year. We eliminated columns that weren't useful for our analysis. Additionally, we did some data

wrangling to convert years into decades. We handled genre and decades separately for our analysis. Our final product was the following dataframe:

	X	title	tag	year	lyrics	decade
1	1422315	Christmas Eve In My Home Town	pop	1966	There's so much to remember No wonder I remember...	1960
2	683066	Those Were the Days	rock	1968	When the city of Atlantis Stood serene above the sea ...	1960
3	578374	Crowded	pop	1968	Crowded, my life's so crowded Not enough time to do...	1960
4	3151856	The Samoan Surfriders - Lau Sei Sili Ese English Trans...	pop	1964	My sei My rose sei My unique treasured sei That was l...	1960
5	1101996	Hey Brother Pour The Wine	pop	1964	Here we sit enjoying the shade Drink the drink that I ...	1960
6	472611	Two of a Kind	pop	1965	They say we're searching for love we'll never find The...	1960
7	3235632	Sally Go Round the Roses	pop	1967	(Sally go 'round the roses) Sally go 'round the roses (S...	1960
8	2370766	Im With You	rb	1967	[Intro] In case you want to know how I feel about you,...	1960
9	1342603	Upstairs Downstairs	pop	1967	Upstairs, mmmm Downstairs, mmmm Upstairs, mmm...	1960
10	4654461	Barefootin	rock	1969	[Intro: Scott Morgan] (Ahh) [Verse 1] Everybody get on...	1960
11	1017920	I Didnt Know What Time It Was	pop	1961	I didn't know what time it was That I met you Oh what...	1960
12	2664975	Thing-a-Ling Scared Crow	pop	1968	Suddenly, she came upon a blinding Yellow-Green lig...	1960
13	1943827	Ive Got That Feeling Bonus Track	rock	1964	I've got that feeling, it's her, oh yeah I've got that feeli...	1960
14	4903895	Blackenized	rb	1969	You fellas better quit jivin'—you better get yourself to...	1960
15	592276	Smile	pop	1961	Smile, though your heart is aching Smile even though...	1960
16	3175472	Golden Arrow	rb	1967	[Intro] Who can shoot the golden arrow? And who can...	1960
17	4186721	Losin My Touch	pop	1965	Baby, something's missing What could it be, I don't k...	1960
18	4343138	Guess Ill Hang My Tears Out To Dry	pop	1964	When I want rain, I get sunny weather I'm just as blue ...	1960
19	4381258	Do You Remember?	pop	1969	I know who I'll run to Shout it out aloud When the eve...	1960
20	758490	Love is a Beautiful Thing	pop	1967	(What you know about love?) I know love is out of sig...	1960

Showing 1 to 21 of 70,000 entries, 6 total columns

Cleaning the Corpus

After cleaning and sorting the data, we had to do additional cleaning on the text corpus (lyrics). To do this process, we used the Text Mining (tm) package in R. We also created a function that combined different functions within the tm package to clean the text corpus. Our function converted our text corpus into a vector and made all words lowercase, kept stem words (words that remain once you remove prefixes/suffixes), and removed numbers, punctuation, English stopwords, and white space.

Building Document Term Matrix

A document term matrix is a matrix that describes the frequency of terms within our data. This matrix is necessary for us to perform statistical analysis on the relationship between lyrics and genre. With this background in mind, we built a function that generated a document term matrix as a data frame. To do this, the function received a corpus and a category as input. In our code, the corpus is the cleaned lyric corpus described in the previous section and the category is genre. Our function then used the tm package to clean the corpus, create a document term matrix, remove sparse terms, and convert the document term matrix into a dataframe. The function then

returns that dataframe along with a given category as output. Here is how the output looks when showing the 10 most common words in songs from the 2010's:

```
sort(colSums(df[df$category==2010,-length(df)]),decreasing = T)[1:10]
```

```
like  know  dont  just   get   got  love   now  yeah   can
22878 18830 16421 16257 15252 14101 12933 12206 11321 10867
```

Results

Confusion matrix and model accuracy of genre for RF and KNN:

```
        rf.y
         country misc  pop  rap   rb rock
  country    1818  134  503   57  316  241
  misc        131 2031  262  172  110  287
  pop         470  199  906  115  527  636
  rap          57   69   78 2375  290   83
  rb          253   57  323  541 1602  192
  rock        486  234  646   89  296 1289
> sum(diag(rf.cm))/sum(rf.cm)
[1] 0.5606154
```

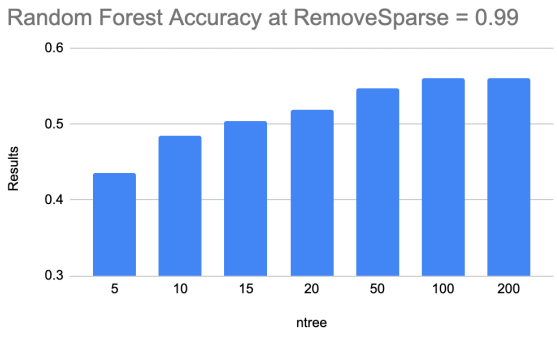
```
        knn.y
         country misc  pop  rap   rb rock
  country     627  985  517  124  233  583
  misc         91 2374  142  127   76  183
  pop         268 1133  506  123  307  516
  rap         123  452  203 1602  337  235
  rb          327  504  458  401  906  372
  rock        217 1278  489   97  195  764
> sum(diag(knn.cm))/sum(knn.cm)
[1] 0.3792448
```

Based on our results, RF was the most accurate prediction tool.

Analysis

Relationship between Lyrics & Genre - Random Forest

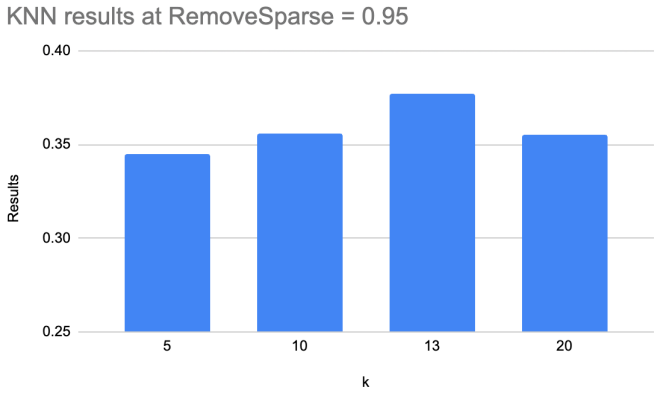
The best result we were able to achieve was 56.06% accuracy with RF. This was with Removing 1% of sparse terms and *ntrees* = 200. We did see that when holding everything else constant, our accuracy increased as *ntrees* increased.



As we got to *ntrees* = 100 or higher, the code took an extremely long time to run and it became impractical to keep going. That, and the fact that we saw very minimal improvement between *ntrees* = 100 and *ntrees* = 200 made us comfortable to stop where we did. We also know that increasing *ntrees* ad infinitum would have led to overfitting and a decrease in accuracy (bias-variance tradeoff).

Relationship between Lyrics & Genre - KNN

KNN overall was significantly worse at prediction than RF. This indicates that the data didn't cluster very well in multidimensional space. The best result we saw from KNN was actually the first run we did, which gave us 37.9% accuracy (recall a random guess would be about right about 16.7% of the time). This run was done after removing 5% of the sparse terms and choosing $k = 13$. KNN also took an extremely long time to run, limiting the amount of runs we were able to do. We did however see that 13 is probably close to the optimal number of neighbors to consider based on the graph below.



Conclusions and Future Direction

The accuracy of the models we chose provided insight to the correlation between song lyrics and genre. Had it been random chance, we'd only be able to predict the respective genre 1/6th of the time. When comparing models, we noticed that RF was more successful with predicting genre and decade compared to KNN. Because RF's accuracy is the average of a handful of different classification trees, we believe that there is a classification tree out there that is more accurate than RF. As for KNN, we learned that the DTM didn't cluster well and a lot of songs spanned

multiple genres according to the word association we had already established. Using these RF provided inspiration for a more accurate model in future endeavors and KNN provided the sobering thought that there is more to genre than only lyrics. The point on KNN was made much more clear when we took the average distributions of words per each genre and plotted six points, one for each genre within KNN and had the algorithm predict based on its closest neighbor. This resulted in our least accurate model, at around 18% accuracy. If there were a true distribution of words among genres in our sample, it would have been these points of genre's mean distribution, but the results yielded were barely above 1/6th. At the most basic level of comparison, we believe that RF was better fit for prediction because it can branch and classify based on single key words specific to genre whereas KNN has to take every word into account.

One more thought to consider about our project and code we used for it. We made the variables general enough to easily adopt other sources of text. It would be interesting to find data that clusters better than vague terms such as 'decade' or 'genre'. Do individual artists have lyrical themes that bleed through over the course of their career? Could we create a matrix to find similar lyrics between artists? Would this then create a new, nuanced, and more naturally occurring clustering of genres easier to predict?

References

Dataset:

https://www.kaggle.com/datasets/carlosgdcj/genius-song-lyrics-with-language-information

Algorithms:

https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/randomForest

https://www.rdocumentation.org/packages/class/versions/7.3-21/topics/knn

Source Code

```
# initialize environment
library(NLP)
library(tm)
library(class)
library(e1071)
library(randomForest)

# load data
song.data<-read.csv("/yourdirectory/songs_uniformbygenre.csv") #Load data - we used 2 different datasets for decade and genre respectively
song.data<-song.data[,c(1,2,3,5,8)] # keeps id, title, lyrics, year, tag
decade<-floor(song.data$year/10)*10 # create decade vector
song.data$decade<-decade # save decade as column
song.data<-subset(song.data,decade>=1960) #cut off songs before 1960 to minimize the number of categories to 7
table(song.data$decade) #data check to make sure we have a uniform distribution
# clean data function using tm
clean <- function(corpus){
  corpus<-VCorpus(VectorSource(corpus)) #create a corpus from the raw data
  corpus<-tm_map(corpus,content_transformer(tolower)) #transform lyrics to lowercase
  corpus<-tm_map(corpus,removeNumbers) #remove numbers from lyrics
  corpus<-tm_map(corpus,removePunctuation) #remove punctuation from lyrics
  corpus<-tm_map(corpus,removeWords, stopwords('en')) # removes common words (the...)
  corpus<-tm_map(corpus,stemDocument) # loved->love
  corpus<-tm_map(corpus,stripWhitespace) #remove excess white space from lyrics
  return(corpus)
}
# build document term matrix
generateDTM <- function(category,corpus){
  corpus<-clean(corpus)
  dtm<-DocumentTermMatrix(corpus) #this makes each unique word into a category and gives the number of times each word appears in each song in a very large 2x2 matrix
  dtm<-removeSparseTerms(dtm,0.95) # removes sparse words. We adjusted this parameter to remove different amounts of words.
  df<-as.data.frame(as.matrix(dtm)) #convert the DTM to a dataframe
  df$category<-category #Create a predictor category (we used decade and genre for this)
  return(df)
```

```
}
# running program, creating df
df<-generateDTM(song.data$decade,song.data$lyrics) #create a dataframe with the DTM and choose a category to predict
sort(colSums(df[df$category==2010,-length(df)]),decreasing = T)[1:10] #show the top 10 most common words in songs from 2010 - we mostly used this as a gut check

# getting ready for models
df$category<-as.factor(df$category) #Convert our category into a factor - this is necessary with decade because it's an integer
set.seed(123) #set seed to make our results replicable
index<-sample(c(T,F),nrow(df),replace=T,prob=c(0.7,0.3)) #Split the data into 70% training and 30% testing
train<-df[index,]
test<-df[!index,]

# randomForest
rf <- randomForest(x=train[-length(train)],
                   y=train$category,
                   ntree=10) #choose the number of trees to use
rf.y<-predict(rf,newdata=test[-length(test)])
rf.cm<-table(test$category,rf.y)
rf.cm
sum(diag(rf.cm))/sum(rf.cm) #divide the diagonal of the confusion matrix bu the sum to get the accuracy of the model

# K Nearest Neighbors
knn.y <- knn(train=train[,-length(train)],
          test=test[,-length(test)],
          cl=train$category,
          k=13) #choose the number of most similar songs  to look at to predict the test data
knn.cm<-table(test$category,knn.y)
knn.cm
sum(diag(knn.cm))/sum(knn.cm) #divide the diagonal of the confusion matrix by the sum to get the accuracy of the model
```