Song Genre Classification Project

Alvito DiStefano

2025-04-15

Introduction

The goal of this project was to determine if song lyrics are a good predictor of song genre. Specifically, this project explored whether song lyrics in bag of words format could be input into a classification model that could predict song genre with any appreciable consistency. To accomplish this, data from the Million Song Dataset (MSD) was utilized. This is among the largest available collections of music data across a wide variety of styles and genres. Two sub-collections within the MSD were used: Last.fm song tags (which included genre information) and musicXmatch (mXm) bag of words lyrics data. Eight genres were extracted from the Last.fm tags for evaluation: indie, rock, punk, metal, rap, pop, blues, and country. These selections were not meant to be mutually exclusive but rather were intended to provide a broad representation of popular music genres to help determine if some genres are more predictable than others. To provide a performance baseline for benchmarking results against, the methodology expounded upon below was first used for spam-ham classification using the SMS Spam Collection, a public set of SMS labeled messages widely used in classification modeling.

Methods

Data Preparation

After automatically downloading the MSD collections, a series of SQL queries are executed using RSQLite to prepare and concatenate the data. Genres are extracted from Last.fm tags using regular expressions formatted like so: (ˆ|[[:space:]-])rap([[:space:]-]|$). This regex is designed to minimize false positives that could impact modeling results. In this example, it picks up on sub-genres like "summertime rap" and "Club-rap" while ignoring tags like "anger therapy" and "trap classics". Track IDs are used to join the mXm lyrics with the corresponding Last.fm genres. Additionally, the Last.fm webpage identifies certain track IDs noted for having matching errors and being unreliable. These are read by the R script and removed from the concatenated mXm–Last.fm table.

Using the regex format described above, columns are created for each genre indicating "Y" if the genre is present in a given track and "N" if it is not. The bag-of-words lyrics are provided already cleaned and stemmed, so minimal cleaning is necessary. The final data.table of combined mXm and Last.fm data is loaded from the R script output as "song_data.RData".

Data Exploration

Taking a closer look at song_data, there are 167,779 rows or tracks in the dataset. There are 5,009 columns, 5,000 of which are word counts from mXm. Function Get_Word_Totals extracts the total counts of each word across all tracks for visualization purposes.

```
##               track_id is_indie is_rock is_country is_blues is_pop is_metal
##                 <char>   <char>  <char>     <char>   <char> <char>   <char>
##   1: TRCCJKN128F426E04D        N       N          N        N      N        N
##   2: TRCCILA128F42BA535        Y       Y          N        N      Y        N
##   3: TRCCHEP128F932DE5F        N       N          N        N      N        N
##   4: TRCCHSQ128F42BB135        N       N          N        N      Y        N
##   5: TRCCOOO128F146368B        N       N          N        Y      Y        N
##   6: TRCCOFQ128F4285A9E        N       Y          N        N      N        N
##   7: TRCCOMD128F421A4CB        N       N          N        N      Y        N
##   8: TRCCMYN12903CB655D        N       N          N        N      N        N
##   9: TRCCMZW128F425C20F        Y       Y          N        N      Y        N
##  10: TRCCRMZ128F42645FE        N       N          N        N      N        N
##      is_punk is_rap     i   the   you    to   and     a    me    it   not   in.
##       <char> <char> <num> <num> <num> <num> <num> <num> <num> <num> <num> <num>
##   1:       N      Y    16    21     2     7    15     6     3    12     6     5
##   2:       N      N    14     5    12     0     4     0     7     2     3     2
##   3:       N      N     1     0     0     0     0     8     7     0     0    12
##   4:       N      N     0     0     0     0     0     4     0     0     0     4
##   5:       N      N     3     2     8     2     3     6     9     1     1     2
##   6:       Y      N     0     0     0     0     0     0     0     0     0     0
##   7:       Y      N     4     5    22     2    10     3     0     4     5     3
##   8:       N      N     2     7    16     5     6     1     0     1     1     4
##   9:       N      N    22    12    15     2    10     2     3     4     1     0
##  10:       N      N     1     9    68     6    24     0     1     1     3     9
```

```r
# Function to count the number of instances each word occurs in the input dataset

Get_Word_Totals <- function(data,start_col) {

  # Create data table to store totals
  word_totals <- as.data.table(matrix(NA_real_,nrow=1,ncol=ncol(data)))

  # Set names equal to song data
  setnames(word_totals,names(data))

  # Sum counts of each word in song data
  for (i in start_col:ncol(data)) {

    word_totals[1,i] <- sum(data[[i]])

  }

  return (word_totals)
}

# Explore how many songs are in the dataset
paste("Total tracks in dataset:",nrow(song_data))
```

```
## [1] "Total tracks in dataset: 167779"
```

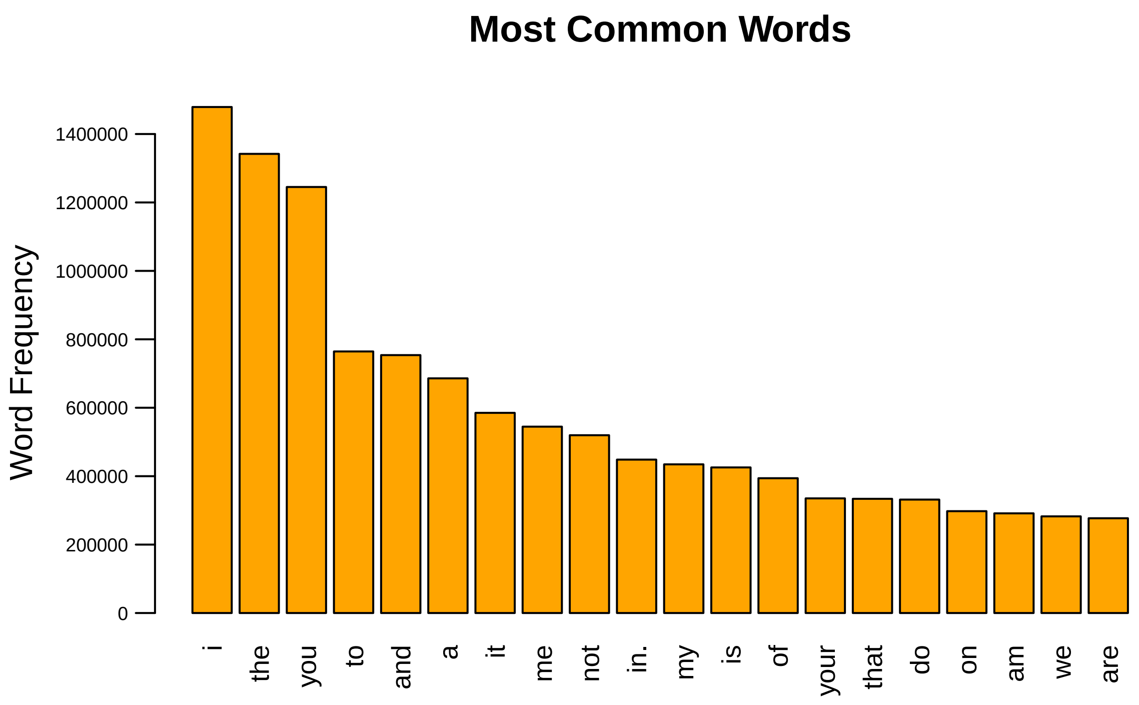
```
# Use function to extract total word appearances as array
word_totals <- as.matrix(Get_Word_Totals(song_data,10))

# Drop NA values
word_totals <- word_totals[,apply(word_totals,2,function(x)!any(is.na(x)))]

# Get indices and sort largest to smallest
word_indices <- order(word_totals,decreasing=TRUE)

word_totals <- word_totals[word_indices]

# Explore word frequencies in song data
paste("The most common word is",names(word_totals)[which.max(word_totals)])
```

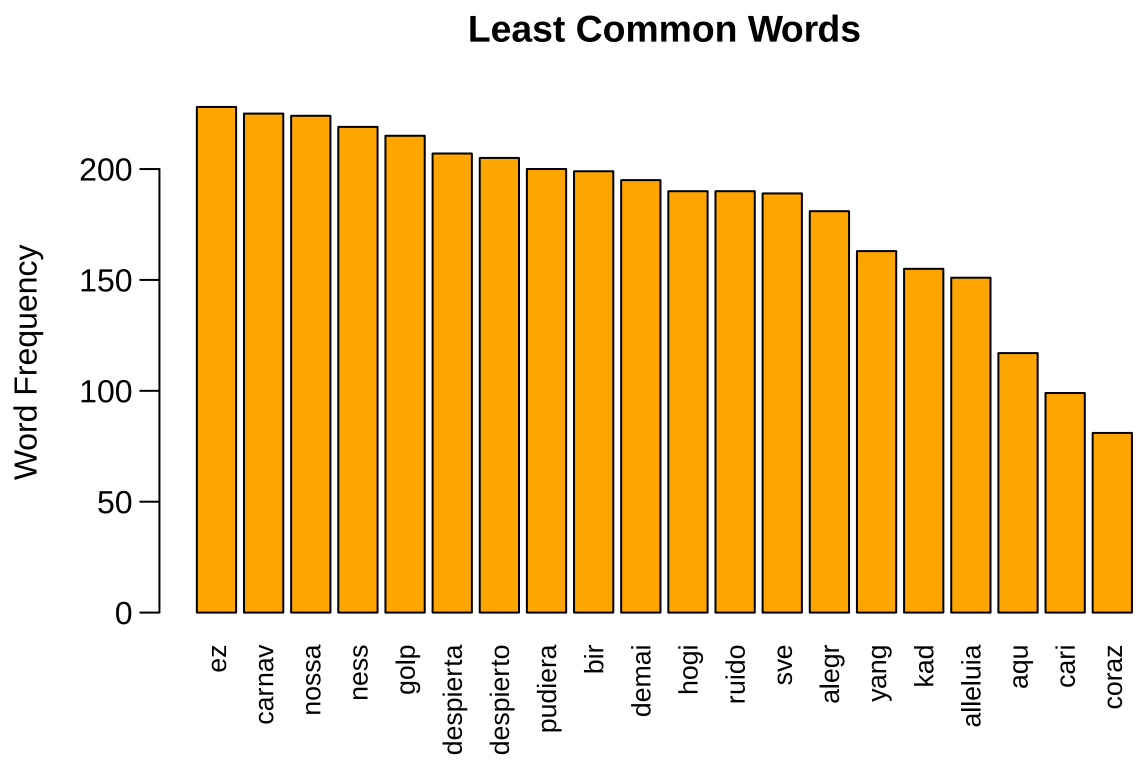
```
## [1] "The most common word is i"
```

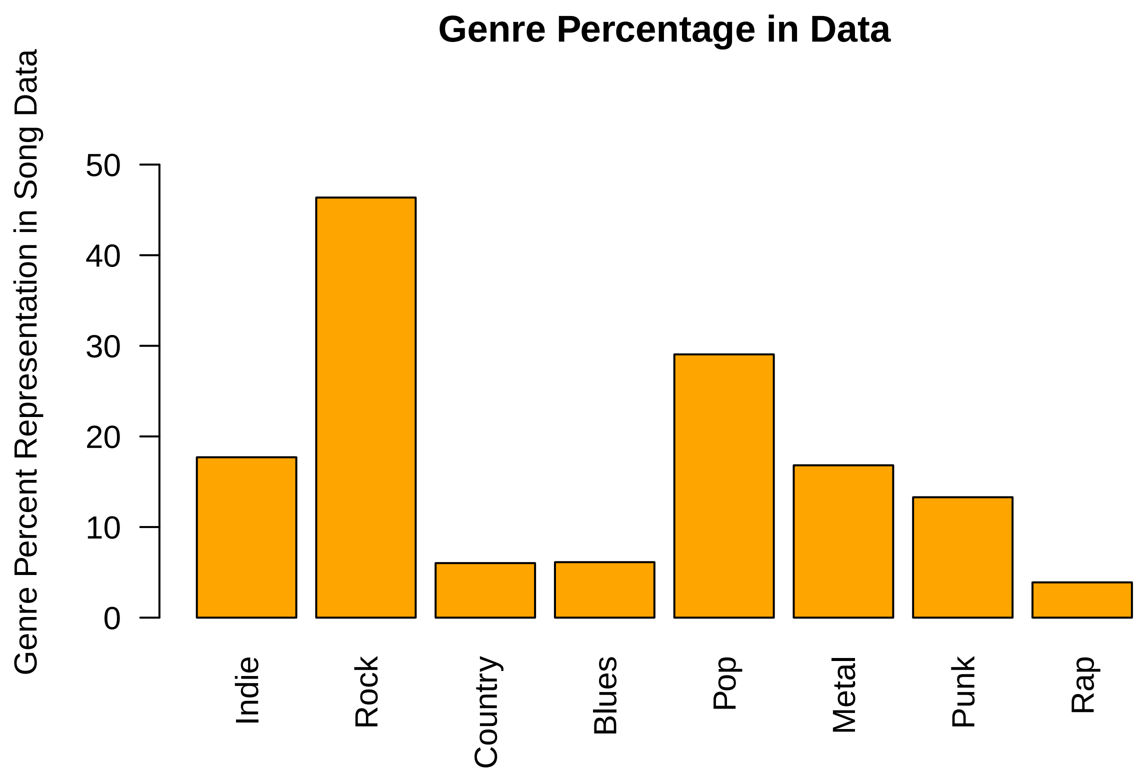
```
paste("The least common word is",names(word_totals)[which.min(word_totals)])
```

```
## [1] "The least common word is coraz"
```

Plotting the most and least common words from word_totals reveals some expected results. Words like "i", "the", and "you" appear over a million times across the ~170 thousand tracks. Conversely, most of the least common words like "coraz", "ruido", and "pudiera" are in a language other than English (in this case Spanish). This is unsurprising since non-English lyrics constitute a minority of tracks in the MSD. A few English words, like "carnival" and "alleluia", also appear among the least common. Additionally, the percent representation of each genre within the dataset is plotted. Rock is the most prevalant, approaching 50% of all tracks in the dataset, while genres like country, blues and rap have below 10% representation (Note that since there is some overlap between genres the total adds up to more than 100%).


Most Common Words
Word Frequency
0
200000
400000
600000
800000
1000000
1200000
1400000
i
the
you
to
and
a
it
me
not
in.
my
is
of
your
that
do
on
am
we
are


Least Common Words
Word Frequency
0
50
100
150
200
ez
carnav
nossa
ness
golp
despierta
despierto
pudiera
bir
demai
hogi
ruido
sve
alegr
yang
kad
alleluia
aqu
cari
coraz


Genre Percentage in Data
Genre Percent Representation in Song Data
0
10
20
30
40
50
Indie
Rock
Country
Blues
Pop
Metal
Punk
Rap

Data Transformations

Data is transformed with a custom Term Frequency–Inverse Document Frequency (TF-IDF) function to diminish the influence of common words during model training. Additionally, because of the large size of the data, dimensionality reduction is performed by dropping columns with near zero variance (NZV). Input parameters to the NZV function are selected to reduce the number of columns (or words) by about two-thirds. Dropping these columns has a negligible impact on training results and significantly expedites model training.

```r
# Function to perform TF-IDF transformation for bag of words data
# If incl_IDF is FALSE then it only performs the TF operation

TF_IDF <- function(data,incl_IDF,start_col) {

  # Create inverse doc frequency term, if desired

  # Capture starting column count
  N_Col <- ncol(data)

  # Calculate IDF if desired
  if(incl_IDF){

    N_Row <- nrow(data)

    dfreq <- sapply(data,function(col) sum(col!=0))

    # Calculate IDF with log base 10
    idf <- log10(N_Row/(dfreq+1))

  }

  # Add column for total word count in each track
  new_col_name <- "total_words"

  # Sum total words in each track, startin with first bag of words column
  data[,(new_col_name):=rowSums(.SD),.SDcols=start_col:N_Col]#(ncol(data)-1)]

  # Convert from absolute word counts to word term-frequencies
  data[,(start_col:N_Col):=
            lapply(.SD,function(x)x/data[[ncol(data)]]),
           .SDcols = start_col:N_Col]

  # Remove totals columns as it is no longer needed
  data[,ncol(data)] <- NULL

  # Multiply term frequencies by IDF factors, if desired
  if(incl_IDF){

  data[,start_col:ncol(data)] <-
    mapply("*",data[,start_col:ncol(data)],
           idf[start_col:length(idf)],
           SIMPLIFY=FALSE)

  }
  return (data)

}


# Function remove data columns with low or zero variance
# Samples data randomly size samp_div for expediency
# nearZeroVar cutoff parameters also input to function

Remove_NZV <- function(data, start_col,samp_div,fcut,ucut) {

  # Ensure data is a dataframe
  setDF(data)

  # Sample portion of data at random for expediency
  idx <- sample(seq_len(nrow(data)),size=floor(nrow(data)/samp_div),replace=FALSE)

  data_sample <- data[idx,start_col:ncol(data)]

  # Use caret's nearZeroVar with specified input parameters
  nZV <- nearZeroVar(data_sample,freqCut = fcut,uniqueCut = ucut)

  # Because data_sample's index is offset from the input data we must adjust accordingly
  offset <- start_col - 1
  nZV <- nZV + offset

  # If zero variance values are found, drop the corresponding columns
  if (length(nZV)>0) {

    data <- data[,-nZV,drop=FALSE]

  }

  return(data)

}
```

```r
# Implement TF-IDF weights by calling the function above

song_data <- TF_IDF(song_data,TRUE,10)
SMS_data <- TF_IDF(SMS_data,TRUE,3)

# Reduce data dimensionality by removing columns with low variance
# Note this is one instance where the song and SMS data are treated differently
# as different inputs to NearZeroVar are necessary to reduce dimensionality
# by an acceptable amount (the goal being to reduce size by roughly two thirds)

song_data <- Remove_NZV(song_data,10,100,50,0.7)
SMS_data <- Remove_NZV(SMS_data,3,10,50,0.1)
```

ML Modeling

After TF-IDF and NZV operations are performed, song and SMS data are split 80/20 into training and testing sets, respectively. To provide a fair comparison between the genres, it is necessary to train each model on the same number of rows. Therefore, the genre with the least number of positive ("Y") rows is selected as the limiting number of rows. This turns out to be rap, with about 5,000 rows or tracks that fall under this genre. This figure becomes the basis of data sampling for all genres and is input into the Create_Genre_Sample function. In this case, the function samples 5,000 positive and 5,000 negative rows of the specified genre, drops the other genre columns, and returns a dataset of 10,000 rows with a 50/50 positive-negative split for training. So each training set (eight total for each genre) is about 10,000 rows by 1,600 columns. This size is not trivial but represents a significant reduction in dimensionality compared to the 167,779 rows and 5,009 columns in the original dataset.

```r
# This function takes an input genre and sample size and samples positive
# cases of that genre of that size and combines it with an equal number
# of negative cases for a more or less 50/50 split

Create_Genre_Sample <- function(data,genre_name,size,genre_col){

  # Split data into Y and N cases by desired genre
  data_Y <- data[data[[genre_name]]=="Y",]
  data_N <- data[data[[genre_name]]=="N",]

  # Randomly sample desired amount of positive/negative cases
  sample_Y <- data_Y[sample(nrow(data_Y),size),]
  sample_N <- data_N[sample(nrow(data_N),size),]

  # Combine into a single data table and shuffle
  data_sample <- rbind(sample_Y,sample_N)
  data_sample <- data_sample[sample(nrow(data_sample)),]

  # Drop all other genre columns except the one at index genre_col
  # which corresponds to genre_name
  drop_cols <- setdiff(2:9,genre_col)
  data_sample <- data_sample[,-drop_cols]

  return(data_sample)

}
```

By default R only uses a single processing core, so the doSnow library was used to engage additional CPU cores. The CPU used has 12 cores with multithreading for 24 total threads, and the R script dynamically detects these threads and uses all but one for model training. Even with all these cores running in parallel model training remained time consuming due to the data size, therefore a simple decision tree (caret's rpart) was selected over more complex models. To validate results, ten-fold cross validation is used with 3 repeats. An rpart model is trained on folds for each genre as well as the SMS data for benchmarking. Kappa is selected as the training metric with a tune length of 10 with the goal of tuning parameters to produce classification models which predict both accurately and better than random chance. With 23 threads in use the training time was about 30 minutes.

```r
CV_folds <- createMultiFolds(genre_data[[Label_name]],k=10,times=3)
CV_control <- trainControl(method="repeatedcv",number=10,repeats=3,index=CV_folds,search="random")
```

```
Genre_Classifier <- train(Label_formula,
                          data=genre_data[,-c(1)], # Drop track ID column
                          method="rpart",
                          metric="Kappa",
                          trControl=CV_control,
                          tuneLength=10)
```

Results

The accuracy, kappa coefficients, precision, recall, and F1 scores are tabulated below for the SMS baseline and track genres.

```
##         Case Accuracy  Kappa Precision Recall F1Score
## SMS Baseline   0.9506 0.7747    0.8296 0.7778  0.8029
##        indie   0.5507 0.1076    0.2288 0.6601  0.3398
##         rock   0.5705 0.1496    0.5321 0.6559  0.5875
##      country   0.6029 0.0777    0.0990 0.7035  0.1736
##        blues   0.6505 0.0946    0.1085 0.6789  0.1870
##          pop   0.5423 0.1100    0.3452 0.6311  0.4463
##        metal   0.6860 0.2185    0.2902 0.6179  0.3949
##         punk   0.6078 0.1162    0.1938 0.6176  0.2950
##          rap   0.8777 0.2765    0.2047 0.7397  0.3207
```

No genre model achieved an accuracy, kappa, or F1 score equal to or greater than those of the SMS baseline, and for most genres these metrics are significantly lower than the baseline. Among the genre cases, accuracy ranges from 0.5423 (pop) to 0.8777 (rap), kappa ranges from 0.0777 (country) to 0.2765 (rap), and F1 scores range from 0.1736 (country) to 0.5875 (rock). The models have relatively high recall in the 0.6 to 0.7 range, but precision is lower across the board indicating a tendency to favor predicting the positive case which results in more false positives than false negatives. No particular genre appears to dramatically outperform others. Rock achieves the best combination of precision (0.5321) and recall (0.6559), resulting in the highest F1 score of any genre, but its kappa of 0.1496 suggests it does not significantly outperform random chance. Conversely, rap achieves a relatively high accuracy and the highest kappa value of any genre studied; however, its kappa is still well below the baseline and its low precision of 0.2047 indicates that, like the other models, the rap model favors positive predictions resulting in many false positives.

Conclusion

Overall, the results indicate that song lyrics are not strong predictors of song genre, at least when represented in bag of words format. While there appears to be some signal present in mXm song lyrics which can yield genre predictions marginally better than random guessing (kappa > 0), this signal is too weak to produce reliable classification models with the methodology employed herein. This is not entirely surprising given that lyrics are generally of secondary importance in songwriting compared to compositional elements like melody, harmony, and timbre. Within a given genre, lyrics can range from poetic to gibberish to fit a musician's style and abilities. Genre is also somewhat amorphous and based on subjective evaluation, and the process used to derive genre labels from Last.fm tags does not necessarily capture the nuance of this. It is perhaps not surprising that genre cannot be predicted from song lyrics alone in the way ham/spam can be predicted from the contents of an SMS message. Still, there is ample room for further exploration into this subject. Other genres could be explored to see if there are any which are more predictable by lyrics than those within the purview of this project. Additionally, given adequate computational resources it would be worthwhile to use both more data and more sophisticated modeling techniques during training. Implementing boosting algorithms, random forests, or neural networks may yield improvements in predictive capabilities.

References

1. Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011.

2. Almeida, T. & Hidalgo, J. (2011). SMS Spam Collection [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C5CC84.