Classify Songs Into Genres

People love music, but not all kinds of music. We divide music up into genres so that we can enjoy songs that are similar. Genres for music are created from many different parts of music, like beat, instruments, and sound. One part that doesn’t seem to be considered as much is the lyrics in a song. I wanted to know if you classify songs based on their lyrics. This could be used to better define genres or if some gernes have a lot of overlap you could set up a system to suggest songs from other genres that the person might like. You could also use it for evil; for instance, instead of suggesting songs that are similar to your added songs into a playlist, other genres may appear that gradually change your taste of music over to an artist that has paid whatever platform to promote their music. That is the only explanation for why pop songs keep showing up in my Disney song playlist.

The dataset I have for this is an odd one. Turns out that the music industry is really big on enforcing copyright ownership. Even though fair use clearly states analysis is an acceptable use of copy righted material. Almost every lyric dataset has been removed from Kaggle and other places where you find data sets. The data set I have is copy of the 380,000+ lyrics from MetroLyrics found on a GitHub that has also been deleted or made far harder to find, and as of two weeks from downloading the files I have not been able to find it again. The dataset has five features: Artist, Song, Genre, Language, and Lyrics. This dataset is not clean and has a lot of issues that needed to be resolved before it could be used.

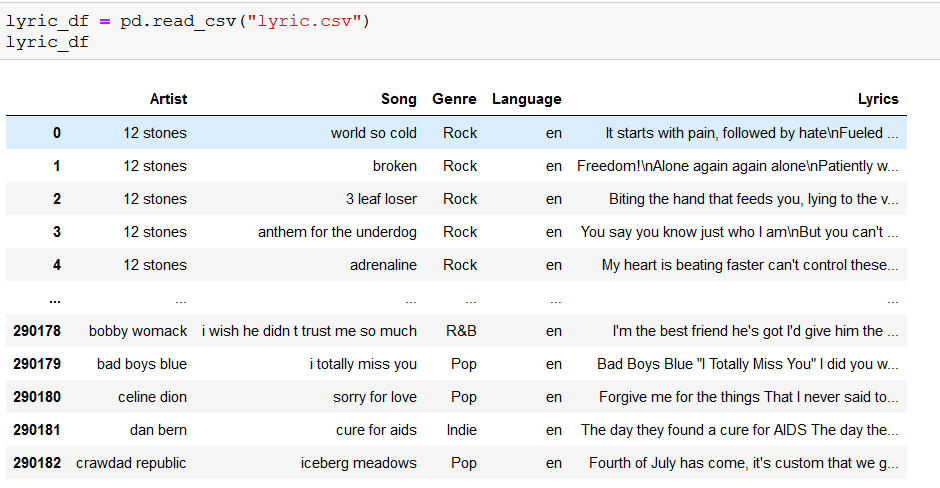
The first thing I did was removed all null values, as there were only thirty-six in the entire dataset. Since I knew I was planning on removing stop words at some point, I filter the data on only songs that were in English. From here I made the decision to remove Language from the features even though the only columns I need are Lyrics and Genres. The Lyrics feature is a string of every word in the song; this is the input feature. To get this ready to be used, many things needed a lot of cleaning. The first thing that needed to be done was removing white space and special characters. I then further filtered the data by only keeping songs that had over five hundred words in the song. The output for this is Gerne, a field with a single word saying which gerne the song belongs to. The gernes that are possible outputs are Rock, Metal, Pop, Indie, Folk, Electronic, R&B, Jazz, Hip-Hop, and Country. However, after running many tests and models, I made the decision to reduce the number of outputs to let the model focus a little more on five genres. I combined some of the songs into the genres of metal into rock, electronic into pop, and folk into country since they seemed to be similar. I then dropped jazz and indie, as those were similar in size and would cut the total genres in half. Near the end of the process, I realized that there were duplicate songs, most likely from different albums that have the same songs. After realizing this, I removed the duplicates; however, that lowered the accuracy of the test as it was no longer getting to cheat with data it already had seen. Before doing anything else to the input data I began to start model selection.

The models that needed to be used were classifiers. For this, I wanted to test a Decision Tree, Multinomial Naïve Bayes, and a Random Forest. However, right off the bat with the over one hundred thousand even after splitting the data into a seventy, thirty, train, test, split a random tree was never going to finish in a reasonable time. After waiting for three hours with no results on the first run of random forest I abandoned the idea of using random forest. The other options that I wanted to explore was which vectorizer would work best in junction with Decision Tree or Multinomial Naïve Bayes. The two vectorizers I wanted to test were TfidfVectorizer and CountVectorizer. After the first run through of the four models with just default settings CountVectorizer Decision Tree had an accuracy of 0.59, TfidfVectorizer Decision Tree had an accuracy of 0.58, CountVectorizer Multinomial Naïve Bayes had an accuracy of 0.53, and TfidfVectorizer Multinomial Naïve Bayes had an accuracy of 0.66. TfidfVectorizer Multinomial Naïve Bayes massively outperformed all the other models. There was an issue that I realized earlier during data cleaning, which is that the data is heavily scud to rock and pop. I ran the same models above on data that was an equal section from each gerne. I think with sinking the data to only a few thousand songs massively dropped the accuracy of all the models into the 0.30s. Thus, I began to tune the TfidfVectorizer Multinomial Naïve Bayes model on the full data. However, due to the size of the data using gridsearch or cross validation it was not returning results in a reasonable time frame. To tune the model, I began to run the models by changing one variable at a time by hand and recording the accuracy. The first test was removing stop words. The results yielded improved accuracy. After that it was changing the ngram\_range, max\_df, and finally the alpha. The final model with the best performance is TfidfVectorizer with ngram\_range=(1,2), and max\_df=0.4 with Multinomial Naïve Bayes at alpha=0.1. There was one more thing I could tune to try and make the model a little better, so I went back to the data cleaning and lemmatizer the lyrics. This is a large change to the data so I reran all the other models to make sure there wasn’t a significant change in the results. While there were improvements across the board it didn’t change which model was performing the best. After all these the changes the final accuracy was 0.69.

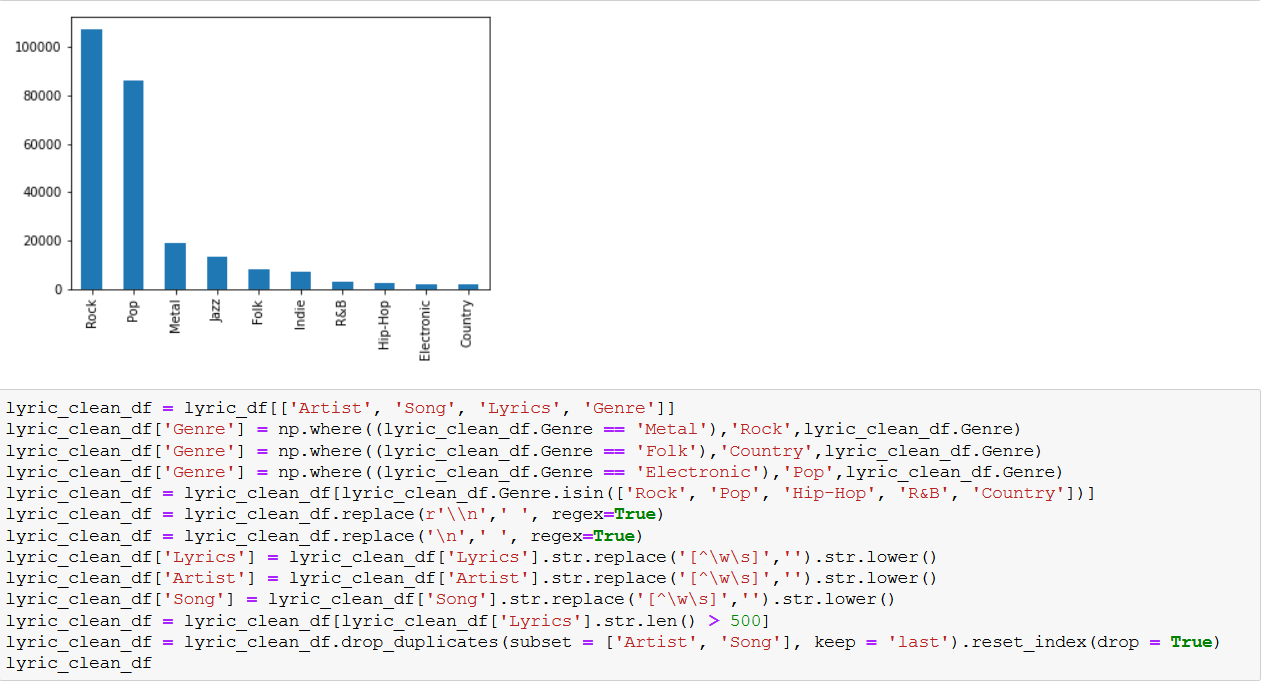
After training and testing the model, I combine the data back into one data set to take a look at what songs were being classified for each genre. Rock and Country both had pretty strong results. However, Hip-Hop had the least amount of pure Hip-Hop and had a lot of Pop mixed into it. From the results of accuracy being 0.69, recall of 0.69, and a precision of 0.71 on the test data in the model I would say we can classify songs by their lyrics.

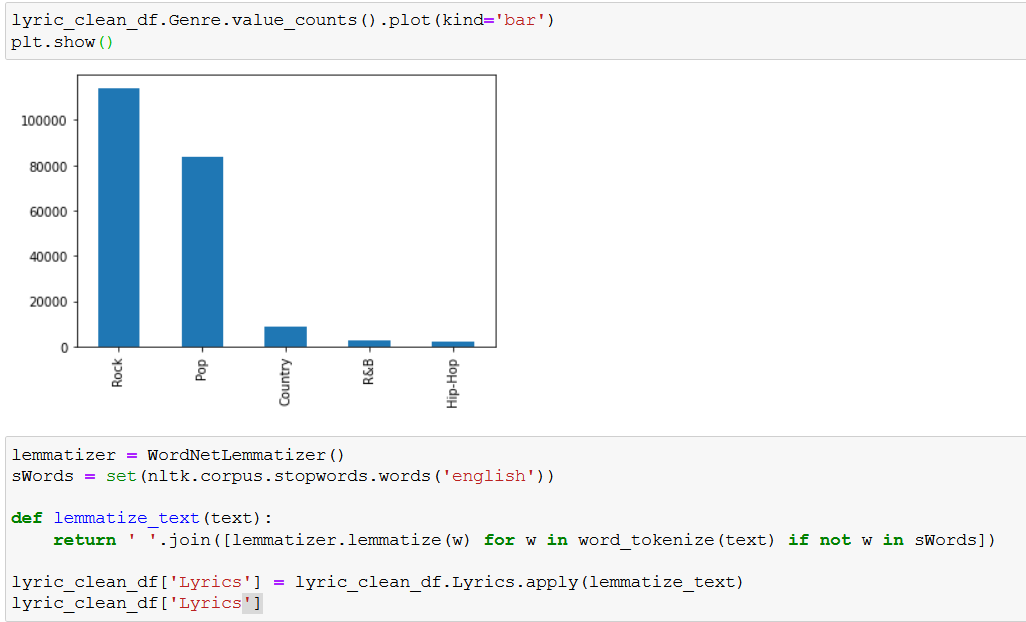
Appendix:

The data

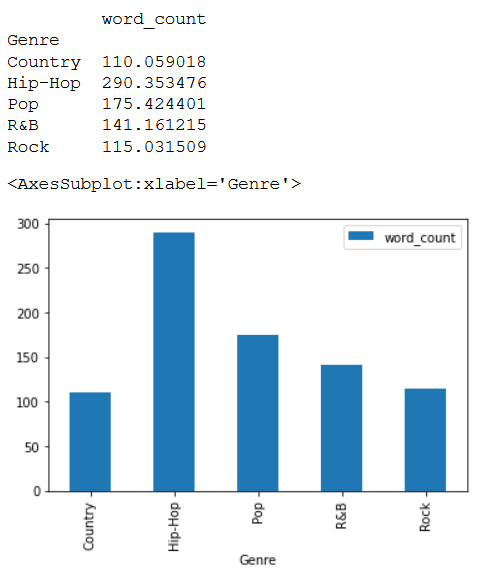


Data cleaning

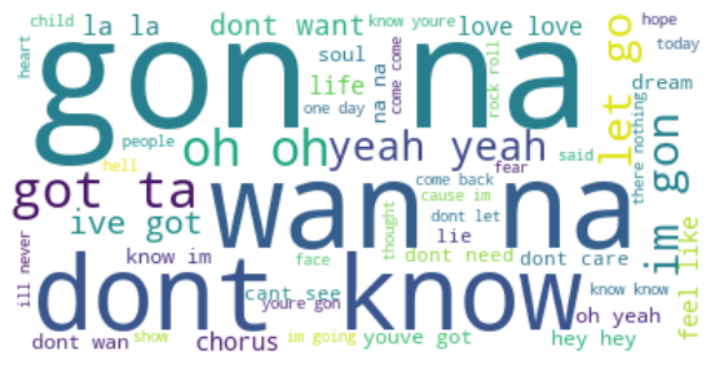




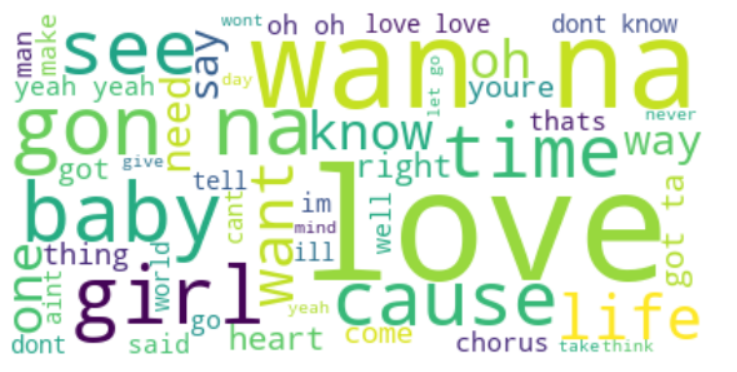
Data exploration



Most common words rock, pop, r&b, country, and hip-hop







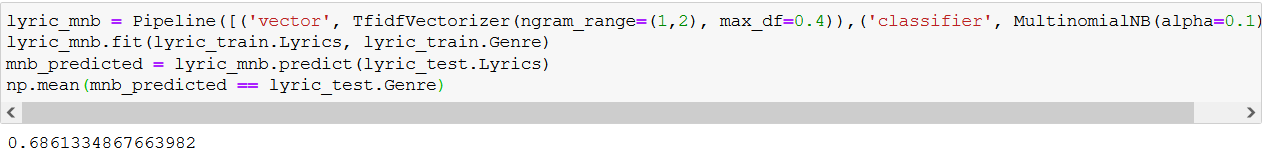


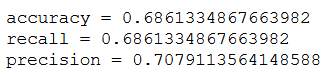


Model building

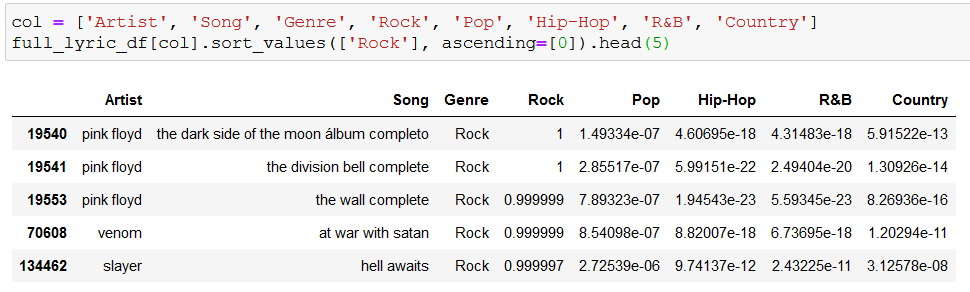


Best model





Top rock songs

Top hip-hop songs

