Music Genre Classification Using Song Lyrics

Chris Lewis

August 10, 2020

1 Introduction

This project is an extension of a project completed for big data programming, which aimed to classify a song's genre based on its lyrics. For the big data implementation, song lyrics were scraped from Genius.com based on pre-determined lists of artists belonging to various genres. The songs were each stored in individual files and were grouped within their genre's directory.

These lyrics were cleaned by removing stop words, non-ascii characters, and lemmatizing them using the NLTK library in Python and stored in a CSV file. PySpark was used to parallelize this process, as the amount of data was fairly large. The CSV file contained columns to organize the songs based on sub-genre, genre groups, cleaned lyrics, sentiment scores, and other points of interest.

For the big data implementation, the cleaned lyrics were used to classify their associated song's genre using stochastic gradient descent using the sklearn Python library. The classes (genres) are as follows:

Class Type	Classes							
Group ID	2			3		1		4
Genre Groups	Rock			Rap		Blues		Country
Sub-Genres	Classic Rock	Punk Rock	Heavy Metal	Pre-2000's Rap	Post-2000's Rap	Blues	R&B	Country

Table 1: Genre groups and sub-genres used as classes

Originally, only the sub-genres were being used to classify the songs, however they proved to be too specific to act as classes, thus they were generalized using groups. The positive and negative sentiment scores were also compared to the accuracies of each predicted genre to identify any correlations between the two, however the sentiment scores were unable to predict the genres themselves.

The extension of this project aims to clean the data further and apply additional machine learning models to improve and compare prediction accuracies. This involved implementing sklearn's multinomial naive Bayes and Bernoulli naive Bayes models in addition to the stochastic gradient descent model. These models had their parameters tuned using sklearn's grid search cross-validation to determine their optimal

values. After tuning both models, they were added to a voting classifier, which uses the model predictions to correct errors made by the three models to generate more accurate predictions.

The original data set generated in the big data programming project can be viewed in the Genres.csv file. The more thoroughly cleaned data set for this project can be viewed in the CleanedGenres.csv file. The cleaned_lyrics column is what will be used to feed into the estimators to make predictions.

2 Results and Discussion

In this section the results for each model are presented and discussed. For each model the parameters were tuned using sklearn's grid-search cross-validation. The impact of changing significant parameters for each model will be also be discussed. Because the data was stored in a CSV file, using the Python library Pandas made generating testing and training sets fairly simple. In order to do so, the Pandas data frame had a randomly selected number row indices selected for a given percentage. For this project the data was randomly divided into 80% training data and 20% testing data.

The sub-genres will not be used for these predictions, as they are far too specific to produce accurate results. In the dataset grouped genres are identified by a unique number. The group numbers and their associated genre names can be seen in table 1, in the group ID row. It is worth noting that the Indie and EDM genres will not be considered in this implementation, as their lyric data is not sufficient to make accurate predictions.

2.1 Pipelines

Before the individual algorithms are discussed, the use of pipelines will be overviewed to explain how the data was prepared before being passed to the models. Sklearn's pipeline module allows the lyric data to be transformed before reaching the final estimator, which will only have to fit the data. Each estimator's pipeline structure is as follows:

$$\text{TfidfVectorizer} \rightarrow \text{Estimator} \rightarrow \text{Fit} \rightarrow \text{Predict}$$

The TfidfVectorizer converts the incoming data to a matrix of TF-IDF features. This is where all important words in each song are weighed, with the TF-IDF value increasing proportionally to the number of times a word appears in a song. The parameters for the TfidfVectorizer were tuned using sklearn's grid-search cross-validation, which performs multiple iterations of a given model, each with a set of different parameters to find the combination that produces the highest accuracy.

The ngram_range and max_df parameters proved to be the most significant in improving each model's accuracy. The ngram_range parameter determines number of items in a contiguous sequence to consider in the song data. The optimal parameter was a range of (1, 2) which considers both unigrams and bigrams when determining the most frequently used words for each genre. The max_df feature informs the vectorizer when to ignore terms that appear in a certain percentage of songs. The

optimal value obtained for this parameter was 0.5. These parameters were used for each estimators' TfidfVectorizer.

2.2 Stochastic Gradient Descent

The overall accuracies and class accuracies calculated using the stochastic gradient descent results are as follows:

Weighted Average	Blues Accuracy	Rock Accuracy	Rap Accuracy	Country Accuracy
0.77	0.70	0.81	0.85	0.56

Table 2: Accuracies using stochastic gradient descent

The plotted confusion matrix calculated using the stochastic gradient descent results is as follows:

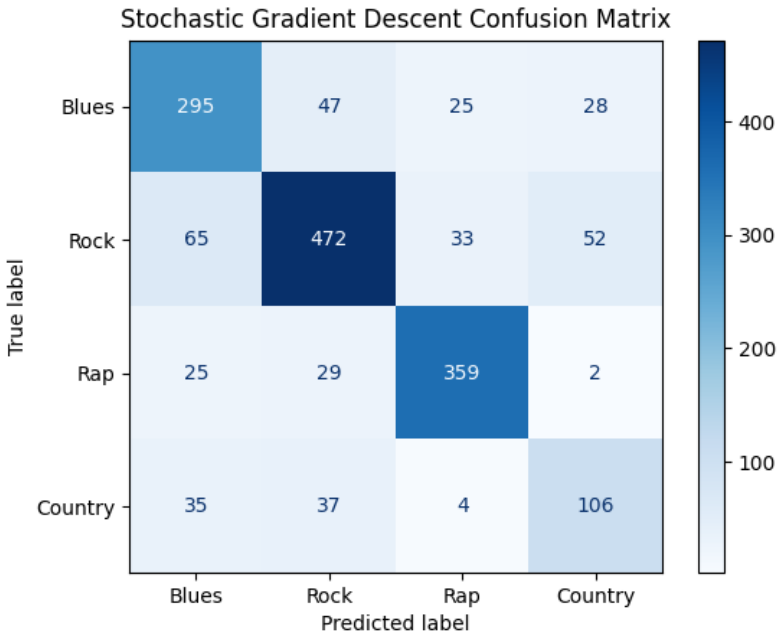

Figure 1: Confusion matrix for stochastic gradient descent predictions

For stochastic gradient descent, the parameters were tuned using grid-search cross-validation. The parameters that were determined to have the largest impact on accuracy were the loss, alpha, max_iter, and class_weight parameters.

For the loss parameter, modified_huber was determined to be the most optimal. According to the sklearn documentation this loss function is equivalent to a quadratically smoothed SVM with a gamma value of 2. This loss function has a higher tolerance to outliers, which are fairly common in song lyrics, therefore it helps bring tolerance to the probability estimates themselves.

The alpha parameter found to be most optimal was the default value of 0.0001, which multiplies the regularization term. There were no significant changes made to this parameter.

The optimal max_iter parameter was determined to be 400, as opposed to the default value of 1000. Because calculating the cost takes a very long time, the grid search determined that 400 iterations is sufficient when reducing the cost.

Finally, the class_weight parameter was determined to be most optimal when set to 'balanced', as opposed to the default value of 'None'. In this case, this parameter is significant because some genres have more songs than others, therefore weighing each class accordingly is beneficial. According to sklearn's documentation, weights are calculated as:

$$N\_Samples/(N\_Classes * Class\_Freq)$$

2.3 Multinomial Naive Bayes

The overall accuracies and class accuracies calculated using multinomial naive Bayes are as follows:

Weighted Average	Blues Accuracy	Rock Accuracy	Rap Accuracy	Country Accuracy
0.78	0.66	0.81	0.85	0.78

Table 3: Accuracies using multinomial naive Bayes

The plotted confusion matrix calculated using multinomial naive Bayes is as follows:

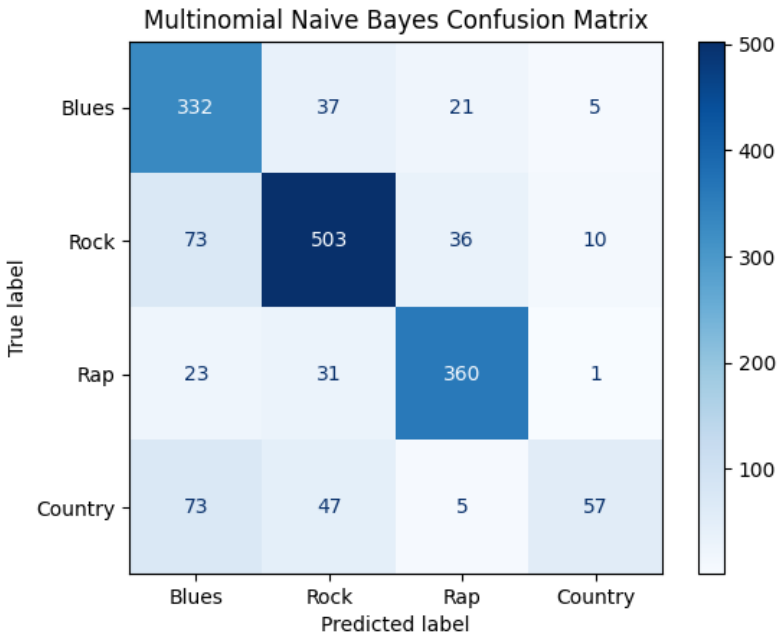

Figure 2: Confusion matrix for multinomial naive Bayes predictions

For multinomial naive Bayes, the parameters that were found to be most significant are the alpha and fit_prior parameters. These were also calculated using sklearn's grid search capabilities.

The optimal alpha value to be used was approximately 0.02864. For this estimator, alpha acts as an additive (Laplace/Lidstone) smoothing parameter, and in this case is much lower than the default value of 1.0. The lower the value, the less smoothing (0 being none), meaning less smoothing is more effective in this implementation.

For the fit_prior parameter, the optimal value found was 'False', as opposed to the default value of 'True'. This parameter refers to when prior probabilities are assigned to training classes or not, which impacts the calculation of the fitted probabilities. When set to 'True', prior probabilities are considered. When set to 'False', the unconditional probability of observing the current (one) class is the same as observing any other class in the data set.

2.4 Bernoulli naive Bayes

The overall accuracies and class accuracies calculated using Bernoulli naive Bayes are as follows:

Weighted Average	Blues Accuracy	Rock Accuracy	Rap Accuracy	Country Accuracy
0.79	0.66	0.85	0.89	0.65

Table 4: Accuracies using Bernoulli naive Bayes

The plotted confusion matrix calculated using Bernoulli naive Bayes is as follows:

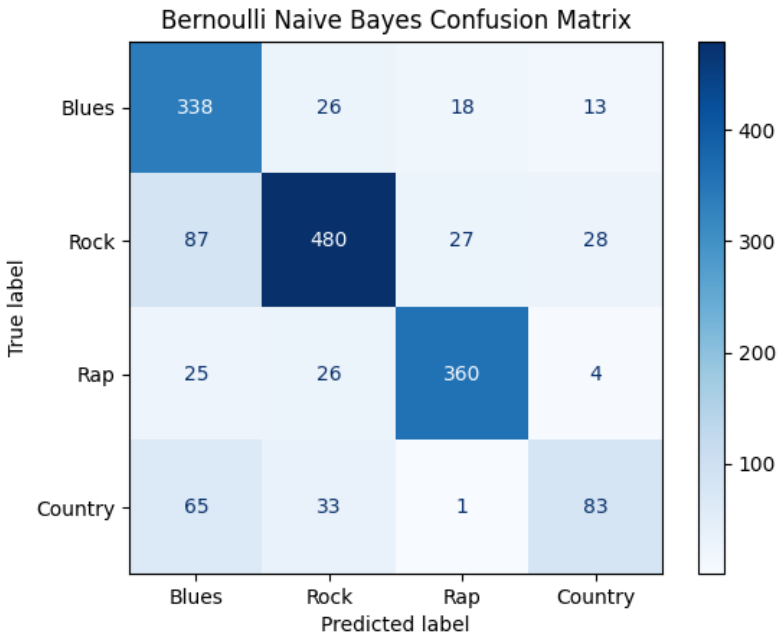

Figure 3: Confusion matrix for Bernoulli naive Bayes predictions

The optimal parameters for the Bernoulli naive Bayes estimator were essentially the same as the multinomial naive Bayes estimator. For example, the optimal alpha value calculated for both models was exactly the same to 18 decimal places.

The only parameter change for this model was in its TfidfVectorizer. In this case, the binary parameter was set to 'True'. Because the Bernoulli naive Bayes estimator models the absence vs. presence of a feature, its TfidfVectorizer must represent its total counts as in a binary (either seen or not seen) format to allow for the Bernoulli implementation to be successful.

2.5 Voting Classifier

The overall accuracies and class accuracies calculated using a voting classifier are as follows:

Weighted Average	Blues Accuracy	Rock Accuracy	Rap Accuracy	Country Accuracy
0.80	0.67	0.84	0.87	0.73

Table 5: Accuracies using a voting classifier

The plotted confusion matrix calculated using a voting classifier is as follows:

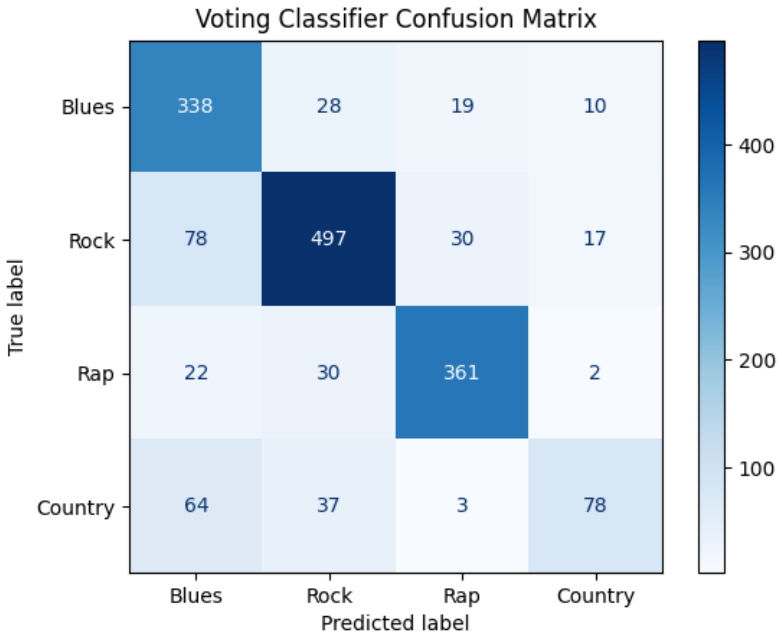

Figure 4: Confusion matrix for voting classifier predictions

The voting classifier combines the stochastic gradient descent, multinomial naive Bayes, and Bernoulli naive Bayes estimators into an ensemble to generate a second level model which perform classification based on the models it contains.

The parameters for the voting classifier were left as is, because it performed best when set to its default parameters. Weights can be provided to this estimator to allow for one model to have a higher impact on the voting classifiers results, however all three models produced results that were very similar, so model weights were unnecessary.

2.6 Probability Analysis

All three models produced very similar overall accuracies, which was highly surprising to me. I expected the stochastic gradient descent and multinomial naive Bayes to have the highest prediction values, as it does not seem intuitive to represent the song data's word frequencies in a binary format, which is used in Bernoulli naive Bayes.

The voting classifier consistently reports higher predictions than any of the individual estimators, meaning one model is not outperforming the others by a significant margin. If this were the case, the averaged accuracies calculated by the voting classifier would be consistently lower than the model reporting significantly more accurate predictions.

It can be seen that the country and blues genres are less accurately predicted. This is due to the fact that, when compared to rock and rap, their songs use less words that are exclusive to only their genre. Rap and rock, on the other hand, use words that are only seen in their respective genres, which directly increases their TF-IDF scores.

3 Conclusions

Overall, the prediction of genre based on song lyrics was improved by using Multinomial naive Bayes, Bernoulli naive Bayes, stochastic gradient descent, and a voting classifier, as opposed to only using stochastic gradient descent. Further cleaning the song lyrics was also beneficial to the classification results. Also, sklearn's parameter tuning abilities using grid-search cross-validation significantly improved the results of each estimator.

In the future, implementing these models to classify songs based on genre would be more effective by finding an online data set that has more thoroughly cleaned song lyrics. The lyrics from Genius.com contained information in the song files, such as who was sining certain verses, which made it difficult to extract meaningful lyric data. As a result, the predictions were negatively impacted. It would also be worth exploring logistic regression as an approach to classify songs, particularly to compare the results to the those of stochastic gradient descent model. All three models performed similarly in terms of prediction and applying a voting classifier allowed the strengths of each model to be utilized.

References

1. "Sklearn.feature_extraction.Text.TfidfVectorizer¶." Scikit, 2020, scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

2. "Sklearn.naive_bayes.MultinomialNB¶." Scikit, 2020, scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html.

3. "Sklearn.linear_model.SGDClassifier¶." Scikit, 2020, scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html.

4. "Sklearn.naive_bayes.BernoulliNB¶." Scikit, 2020, scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html.

5. "Sklearn.ensemble.VotingClassifier¶." Scikit, 2020, scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html.