Lyric Based Classification of Music Genres

Austin Cook
CS 270, Fall 2024
Department of Computer Science
Brigham Young University
ac898@byu.edu

Abstract

The purpose of this project is to determine if music genre can be classified from lyrics alone. Music is a large industry around the world, and advanced machine learning algorithms are employed to improve user experience by recommending songs and artists, supporting autoplay, and creating customized playlists. While these algorithms are typically developed by teams of machine learning experts, the goal here is to create a set of simple models that create groundwork for understanding how more advanced algorithms could be created. To solve this problem, a custom dataset was created, with lyrics pulled from Spotify, and metrics extracted from the lyrics. Each model was trained on the dataset without normalization, with normalization, and with further model tweaks for optimization. While test accuracy capped out at about 80% with the most successful model, a significant improvement is demonstrated over the baseline accuracy of 33%. Not all songs are easily discernable from lyrics alone, with overlap existing between genres. Future work could branch into exploring additional lyric features as well as analyzing audio for musical elements beyond lyrics alone.

1 Introduction

One of the great pleasures in life that most can relate to is music. Each person has a unique set of preferences, and it has never been easier than now to access a seemingly endless amount of music. With the popularization of music streaming services allowing access to virtually all music without the requirement to purchase each song, individuals are able to listen to music without financial and access limitations common in the past. Platforms such as Spotify expend great resources to tailor music recommendations to users. This includes the employment of machine learning algorithms to classify music, identify trends, cluster songs, and analyze user listening history.

The goal of this project is to determine whether music genres can be classified from metrics derived from lyrics alone. Successfully reaching this goal required two things. First, proper features needed to be identified. The idea of feature selection is to select key properties that help to differentiate between instances of one class versus another. Second, effective models and hyperparameters for each needed to be selected. Inductive bias is how a given model generalizes on data. While some inductive biases may work for a problem, it is also guaranteed that others will not work as well. Through experimentation, I discover which inductive biases are more effective for solving the problem of music classification.

In order to keep the project within an appropriate time and complexity scope, I decided to focus on classification between three genres: country, pop, and contemporary Christian. These genres were selected for a few key reasons. First, while exceptions exist, songs in each genre tend to have lyrics. Choosing another genre that doesn't contain words would effectively diminish the purpose of this project because the focus is identification from lyrics. Secondly, I chose these genres for the readily available playlists containing large numbers of popular songs in each genre. A goal with data collection was to have a fair representation of songs across each genre rather than a niche within the genre, which is less likely to happen with larger playlists of popular songs.

While the methods of this project are simple in comparison to the complex application of algorithms employed by large music streaming services, the results are a concrete example of how songs can be selected to tailor to individuals with particular genre preferences.

2 Methods

2.1 Feature Selection

The goal in feature selection was to identify which aspects of the data would distinguish genre. I thought of some simple ones at first such as word count, average line length, word variety and profanity. Additionally, I realized that the words themselves probably tell a lot about the genre. For example, in contemporary Christian songs tend to focus on faith, using words related to faith and religion. Country songs tend to focus on themes such as love, work, conservative values, and rural life. Given this, I decided to analyze the frequency of typical words from each genre, giving care to not overfit to words that appear only a few times within a genre, while not appearing in other genres. Three features were added, one for common words in each genre, respectively.

Table 1: Feature Descriptions

Feature Name	Feature Description
word_count	Total number of words in the song
avg_line_len	Average length of each line
word_variety	Percentage of words that are unique
profanity_freq	Percentage of words that are profane
country_typ_words	# of words commonly in country
pop_typ_words	# of words commonly in pop
christian_typ_words	# of words commonly in Christian

2.2 Data Source

2.2.1 Gathering Song Lyrics

Data collection was a significant portion of this project. While large datasets of song lyrics are available online, I wanted to be able to easily pull song lyrics from a given list. Additionally, much of the data preparation involved metric extraction, which was more difficult to find in a ready-made dataset.

The first step to data collection was to find lists of songs to use in analysis. I found large playlists of 1500 songs or more on Spotify for each of the three genres in this project. These playlists were selected due to their large number of songs and fairly representative coverage of each genre. To pull the track lyrics for each song in each playlist, I explored various potential APIs. Spotify uses Musixmatch, a third-party company, to provide lyrics to display in their app. Musixmatch and Spotify both use corresponding track ids. Musixmatch, however, requires a developer account, which is not open for anyone to create. I decided to use an API called SpotifyScraper. This API acts as a proxy between its subscribers and Musixmatch, allowing access without requiring a developer API key.

Next, I sent each playlist id to the SpotifyScraper API with a request for all track ids. Since the API will only return the first 100 from a playlist, I broke each down into groups of 100, and then collected track information and parsed the track ids from the JSON. Next, I pulled the lyrics for each song and combined them all into a single CSV file including the title, first artist, track_id, lyrics and genre.

2.2.2 Creating Genre Typical Word Lists

Since three of the metrics were typical words, I wrote a program to analyze the lyrics from each genre to derive the lists. Note that these words were derived from the same dataset as used for training and testing. While this could potentially cause overfit, I proceeded due to time constraints, having already completed this process when I realized the potential issues. However, I employed methods to help ensure that sufficient generalization was achieved. At first, I only included words that were exclusive to each genre. The issue with this, however, is that it is essentially overfitting for each genre. When any genre exclusive word appeared in a song, it was essentially giving away the classification with a hardcoded answer and was not generalized to work well with all music.

Instead, I decided to keep the top 200 most common words from each genre not appearing more than 500 times in either other genre. Only keeping 200 words from each genre helps to avoid overfit by using words that are general to many songs of the respective genre. Leaving out words appearing more than 500 times in either other genre helped eliminate words that wouldn't help distinguish between genres. Before this process, the words were also run through a list of stopwords, to remove those that don't add much value. These are common words such as 'I', 'you', and 'they'.

2.2.3 Deriving Metrics for the Final Dataset

The next step was to analyze the lyric data and create a new CSV file with these metrics. I wrote a program to read in song lyrics, one song at a time and process the text to derive the metrics. Using a comprehensive public list of known profanity words, I calculated the profanity_freq of each song. I followed the same process for each list of genre typical words. Each list of metrics was saved as a row item on a CSV file representing the final dataset.

Since not all songs contained lyrics, I dropped those instances, keeping 1400 samples from each genre, with a total of 4200 samples in total.

2.4 Selected Models

Six models were selected to explore the research question. These include Perceptron, Multilayer Perceptron (MLP), K-Nearest Neighbor (KNN), Decision Tree (DT), Random Forest (RF), and Gaussian Naïve Bayes (GaussianNB).

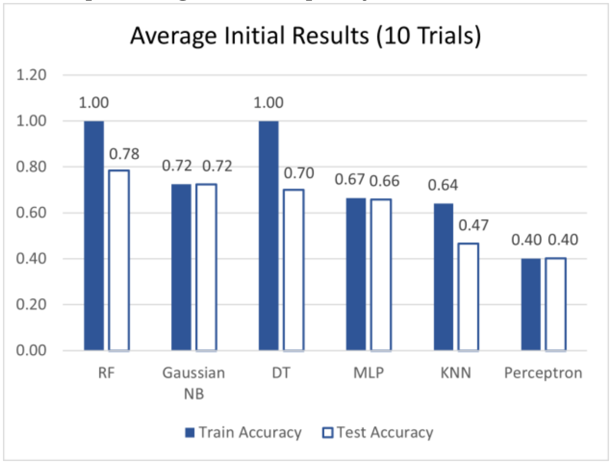
Perceptron was selected as simply a baseline to check the performance of other models, and to explore the linear separability of the dataset. MLP was selected due to its robustness and ability to find complex features. All others were selected due to their variety of approaches and inducive biases in generalizing novel data. The intention here is to find which inductive biases work well with the dataset. RF is a more robust ensemble version of DT, and was selected with the suspicion that it would outperform DT.

3 Initial Results

Scikit-Learn adaptations of each algorithm were used. For initial model testing, I split the data into splits of 80-20 training-to-test data. Defaults hyperparameters were used for each model. Each model was trained 10 times and average training, and test scores were calculated. The goal here was to get a baseline of how well each model would perform without tweaks and improvements.

These results demonstrate the out-of-the-box performance of each model on this particular data set. It is important to note that each model generalizes based on a specific inductive bias. The inductive bias is a set of assumptions that are made by the model which determines the way it will make predictions based on given data. I included the perceptron, because I was curious to see how linearly separable the results are.

Given the fact that it scored about 40% on both training and test data, and the baseline accuracy is 33%, few samples could be effectively split linearly. While KNN appears to perform significantly better than average with training data, like the Perceptron, it generalized poorly with test data.

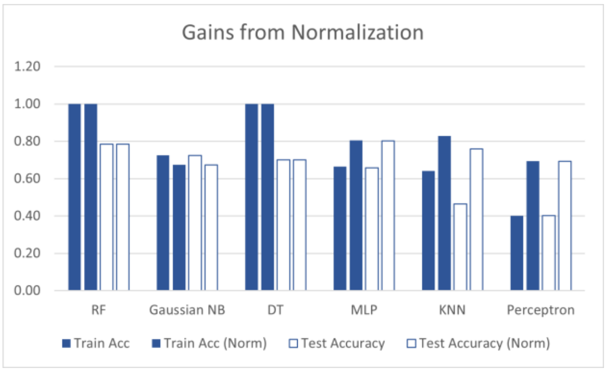


Each other model performed significantly better. The robust set of trials in random forest gave it an 8% increase over single decision trees. NB worked well out of the box. Since, in my experience, hyperparameter selection tends to be important and impactful with MLP, I predicted that further adjustments would benefit it more than other models.

4 Improvements

4.1 Normalization

I first experimented with normalization to see which models would respond more to having a standardized distribution of the data. It was interesting to find out that some models benefited greatly from normalization. Perceptron and KNN both saw the most benefit, at a 29% increase in test accuracy. I hypothesize that Perceptron increased so much due to its inductive bias of finding a single line through all points of the data. Normalizing the data would help put the data in a plane where it would be easier to find such a line. KNN relies on calculating the distance between samples to determine which are closer. Without normalizing the data, features that tend to have larger values than others, even if they aren't less important, will skew points to seem like they are further apart than they really are.



MLP saw a 14% increase in test accuracy, about half that of Perceptron and KNN, though still significant. Again, I hypothesize that the improvement has to do with normalization making weights start out in a more standardized fashion.

As expected, DT and RF didn't improve at all. Both algorithms rely on splitting samples into subsets based on feature values, making a tree. Since the splits are determined by finding the most efficient attribute to split on, based on lowering entropy, the scale of the values is arbitrary.

Perhaps the most interesting observation is that GaussianNB had a decrease in performance due to normalization. What I learned is that GaussianNB depends on data being roughly in a Gaussian distribution to work well. Normalization can sometimes disrupt an already effective distribution leading to worse results.

4.2 Model Tweaks

For all algorithms other than GaussianNB, I normalized the data. GaussianNB was the only algorithm to have lower performance with NB, and I chose to leave data unnormalized for GaussianNB to give it the best chance of significant improvement, while normalizing data for the others for the same reason.

Perceptron
Being a fairly simple algorithm, perceptron doesn't have many hyperparameters that can be manipulated to improve results. Experimenting with the learning rate, I was able to improve results by 4%. With learning rates of 0.001, and 0.01, test accuracy was the highest, after which increasing it only decreased accuracy.

Multilayer Perceptron with Backpropagation (MLP)
As explained above, MLP is a highly adaptable algorithm, and has a large amount of hyperparameters that can be adjusted. For this algorithm, I decided to use Scikit-Learn's Grid Search to experiment with parameters. Grid Search works by accepting an object with various values for hyperparameters with which you would like to experiment. Due to avoiding an excessively large search space, I chose three hyperparameters to experiment with, leaving others constant.

The three hyperparameters that were adjusted were learning rate, hidden layer sizes, and momentum. For learning rate, I tried 0.01, 0.1, and 1. For hidden layer sizes, I tried 32, 64, and 128. For momentum, I tried 0, 0.25, and 0.5. Interestingly, the best result didn't beat the default values of learning rate of 0.0001, 100 hidden layers, and 0.9 for momentum. While this happens to be the case with this dataset, it isn't always.

K-Nearest Neighbor (KNN)
For KNN, I experimented with the number of neighbors trying values from 1 to 200. KNN works by classifying a point simply as the most common classification of its neighbors. The number of neighbors sets how many of the closest points to consider. With each value, I again performed 10 trials, averaging the accuracy of each to get a more stable training and test score. Performance peaked at 20 neighbors, yielding a test accuracy of 77.7%, up 1.7% from KNN with the default

of 5 nearest neighbors. Beyond 20 nodes, performance remains high but never improves.

Decision Tree (DT)
To improve the decision tree, I experimented with an iterative approach do discover the best hyperparameters. I selected three hyperparameters that I suspected would be meaningful, then one at a time found the best value. When moving on to the next hyperparameter, I used the best value for the previous hyperparameter. Again, I performed ten trials for each hyperparameter value to get a more stable average than just a single trial.

The three that I experimented with were criterion, min_samples_split, and max_depth. Criterion specifies which algorithm to use to determine how good a split it. The options on Scikit-Learn's DecisionTreeClassifier are gini, entropy, and log_loss. Gini is the default and scored the highest with a training score of 70%, being the same as the normalization test above.

Min_samples_split is a technique that can help avoid overfit. It specifies the minimum number of samples to be in a child node of a split to be able to make the split. When there is no minimum, the training score will always be 100% because the model will be perfectly fit to it. However, that isn't an indication of how well the test set will perform. I experimented with values from 2 to 100 and found significant improvement. 50 was the best value with an average test score of 74.36%, up 4% accuracy from the default.

Max_depth specifies the maximum number of levels the tree structure can have. Again, it is an overfit avoidance technique. Here, I experimented with values from 2 to 200 and finally with no limit. I found that the max depth was never more than 15. The best max depth was 10, with a test score of 74.58%, only slightly better than the default of no depth limit.

This method of incrementally finding the best value of a selected hyperparameter, using the best in the future, and then trying other hyperparameters is my preferred method of hyperparameter experimentation. While grid and random search can experiment with a variety of hyperparameters in an automated way, this method gives the ability to see the impact of each hyperparameter and may save time in the long run due to a smaller search space. Rather than trying all possible combinations against each other, the best combination of each is found one at a time, leading to a reduction in time complexity from n^d to n*d where n is the number of values to test per hyperparameter, and d is the number of hyperparameters to test.

Random Forest (RF)
For RF, I experimented with the same hyperparameters as with DT, criterion, min_samples_split, and max_depth. Interestingly, here entropy and log_loss both beat gini, though only by 0.2%. I used entropy for the selection of the other hyperparameters.

With min_samples_split, test accuracy peaked at 10, while with DT, it peaked at 50. This doesn't come as a surprise because RF creates many decision trees, helping avoid the overfit that DT tree is more likely to have with that few minimum samples per split.

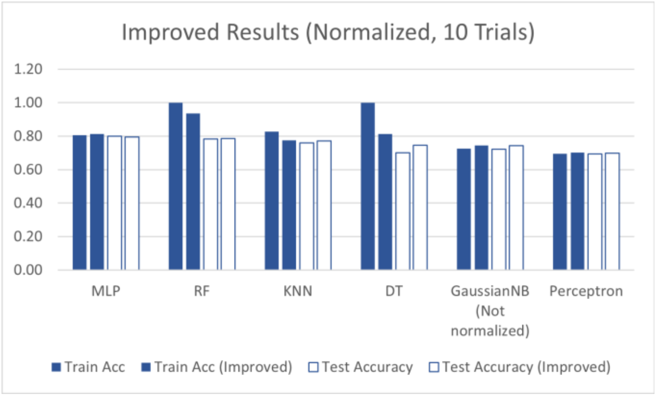
Testing with various values of max_depth, the optimal values was 11, which is very close to the 10 which was optimal for DT. While these refinements helped some, the total improvement was only about 0.2% beyond using the default parameters.

Gaussian Naïve Bayes (NB)
NB is a fairly simple algorithm and doesn't offer a lot of room for refinement through hyperparameters. However, I experimented with the var_smoothing hyperparameter. This adds value to the variance of each feature which helps to stabilize the results. I tried values from 1e-10 to 1e-5. Interestingly, this hyperparameter did lead to an improvement in accuracy. The optimal value was 1.5e-8, yielding a test accuracy of 74.35%, up from 72.32% with the default value of 1e-9.

5 Results and Discussion

Results are ordered by final improved training accuracy. As expected, MLP performed better than all other algorithms on test set accuracy. Due to the ability to use many layers and node, MLP is able to learn complex relationships. Something that I noticed while compiling the dataset was that the value for a specific genre's words didn't always indicate the classification. Rather, it was often the number of words from each genre relative to that of other genres. While the complexity of this dataset is fairly low and straightforward, MLP is able to learn much more complex relationships.

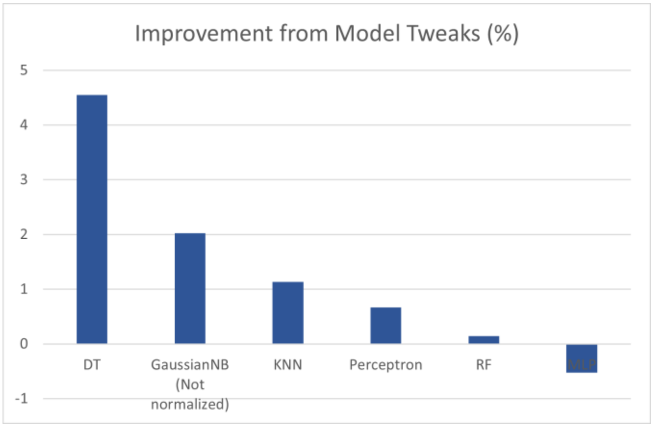
Interestingly, all models, even perceptron, scored upward of 69%. The variation of performance between algorithms is about 11%. The fact that perceptron scored so well speaks to the idea that the dataset is fairly simple. Perceptron is only able to discover linearly separable relationships and scoring that high demonstrates that those relationships exist in the data.



Baseline accuracy is the expected accuracy at a random guess. Because there are three possible classes, with an equal number of samples from each class, the baseline accuracy is 0.33%. With all algorithms scoring over double that, significant success as genre classification was achieved.

It is also notable that no algorithms exceed 80% test accuracy by much. While a large majority of the data is easy to distinguish, I hypothesize that the 20% of samples that are missed by each algorithm have a fairly large overlap due to songs that are simply difficult to distinguish due to lyrics alone. While the features were selected intentionally, songs within a genre can vary widely, and some songs are fairly different from other songs of their genre, making them hard to classify. For this project, features were derived entirely from the lyrics, which have inherent limitations. For example, a country song may have very similar lyrics to a pop song with other musical elements such as melody, rhythm, and pitch are entirely different. Adding such features would help identify genre in songs where lyrics tell little.

Some algorithms benefited more than others from model tweaks. In MLP, for example, I was unable to find a set of hyperparameters which performed better than the defaults. DT, on the other hand, had close to a 5% increase, narrowing the gap between it and RF to less than 4%, despite RF being a very robust algorithm.



GaussianNB came second for improvement from model tweaks. While only beating Perceptron at test accuracy, the model tweaks brought it within 8% of MLP. RF didn't see much improvement, and I suspect the reason is due to the robustness of using so many trials with unique datasets derived from the original dataset. Due to that, its initial performance was already very high, leaving little room for improvement.

Seeing how some algorithms benefited more from refinement than others, it is clear that in situations with limited resources, and little need for highly optimized accuracy, using default hyperparameters can be a good option.

6 Conclusion

In this project, a dataset was created from scratch for use in music genre classification. 4200 songs were pulled from Spotify, custom feature metrics were created, and lyrics were analyzed to extract metrics for input to each algorithm. Due to the extensive work of writing a program to analyze lyrics, dataset creation was a large portion of this project.

Each algorithm was initially run on the unnormalized data dataset and using default hyperparameters. This created a base to compare against when making further modification to the algorithms. Next, the dataset was normalized and run on each algorithm. While RF and DT saw no improvement in accuracy, and GaussianNB saw a reduction in accuracy, the other three algorithms saw a significant improvement.

Next, each model was intentionally refined by training with a variety of hyperparameters. Various methods of model refinement were employed, including grid search and incrementally tuning hyperparameters one at a time.

Results showed a maximum correct classification rate of just over 80% by an MLP. RF and KNN followed closely , with a 1% and 3% drop in accuracy, respectively. All other algorithms followed behind with the least accurate algorithm performing at 70% accuracy.

7 Future Work

Beneficial future work with classifying country, pop, and Christian music would include further feature consideration, as well as the employment of additional ensemble methods, beyond RF. Features were selected intentionally and were derived from lyrics. However, more work could be put into fine tuning those features. Principal component analysis could lead to insights as to which features could be combined due to covariance. This information could help determine what other aspects of lyrics are useful in distinguishing between genres.

Ensemble methods can be a powerful means of classifying due to improving on an algorithm itself or combining it with others to improve results. Each model relies on an inductive bias to generalize for novel data. Data can potentially follow more than one inductive bias, benefiting from the combination of multiple algorithms.

Beyond modification to the current research question of classification between three music genres, a wider view could be taken by exploring more difficult genre classification tasks. More genres could be included, including those with more nuanced differences than the often-stark differences seen between country, pop, and Christian music.

Another potential future research question could be the viability of clustering for song recommendations. For example, Spotify has an autoplay feature where, after finishing all songs from a particular playlist, additional songs similar to those previously listened to are selected and automatically played. Clustering can find groups of related songs for intelligent grouping, without a need to explicitly label each song by its genre.

Beyond lyrics, other musical elements could also be analyzed. As noted above, not all songs can easily be classified by lyrics alone. Analyzing features in the audio would likely be a useful means for classification and clustering.