Song Lyric Classification and Analysis

Michael Brown

Abstract

For a new music service, being able to analyze and classify lyric data by artist and genre is important when that metadata is not given for an uploaded song. It is also important to be able to generate recommendations in order to keep users engaged and expand their listening habits. This project attempts to accomplish both of these tasks using machine learning models for classification and word embedding similarities for recommendations. The result is a classification model capable of predicting genre with a 71% accuracy and artist with a 16% accuracy, alongside a recommendation system that generates diverse song recommendations related in topic to the specified song and artist. In this report, the ways these objectives were completed as well as their detailed results are described.

1 Introduction

Large music streaming apps such as Spotify or Apple Music often have playlist recommendations of songs by genre. However, when a song genre or artist is not specified in song metadata for a newly uploaded song, it would be challenging and time consuming to identify them manually. My project focuses on this issue and develops a model to be able to predict both genre and artist when given the lyrics to a song.

In addition, these music streaming apps like Spotify and Apple Music often have built in systems for recommending new artists and songs to their users based on the listening habits of related users through trends, as well as determining which songs users skip or listen to the most (Spotify). This sometimes results in only the most popular or the same songs being recommended over and over, and may limit smaller artists from being given exposure. In addition, for a new company wanting to generate recommendations, without the necessary large amounts of user data, it would be difficult to create recommendations from trends or listener preferences. As a solution, I wanted to attempt to develop a recommendation system that allows for a chosen number of recommendations based on related song lyrics.

My project will focus on building a system to answer the following questions about the above topics:

1. Is it possible to accurately predict the artist and genre given solely a song's lyrics?

2. Which machine learning model will best accomplish this task, and how will it compare to a baseline model

3. How will recommendations generated from song lyric data compare to recommendations generated by large music streaming apps like Spotify and Apple Music?

2 Related works

When researching for the project, I found a related article which attempts to classify a song into its genre, however instead of using the raw lyric text data, they use a .wav audio representation of the song in order to predict the genre (Agrawal). I found this project to be very interesting since I have only ever worked with analyzing text and numerical data, and transforming audio into numerical data for song classification sounded really neat. The article also details how they decided on K-Nearest Neighbors as the model to use for this classification task, which was also what I had initially planned to use as well due to its ability to classify data based on nearby points.

I also did research into the methodology Spotify uses to generate user recommendations, which consists of generating a "taste profile" based on a user's most listened to artists, song trends from similar users, and most popular current songs (Spotify). This methodology will be different from what I use to generate recommendations, since I will be using just the raw song lyrics, but it is still interesting to see how Spotify curates recommendations, and how it is largely dependent on existing users to generate song data like song popularity & trends.

3 Data

This project will use data from the dataset *Genius Song Lyrics: Language annotated song and lyrics data* by CarlosGDCJ (CarlosGDCJ).

This dataset is a .csv made up of song data scraped from Genius.com, a website containing millions of unique songs and their lyrics. Each of the 5 million data points in the dataset contains the song title, song tag/genre, song artist, song release year, number of page views for the song on Genius, featured artists, the song lyrics as formatted on Genius, and song language info. The 5 features of the dataset I will use in my project will be: song title, song tag, song artist, views, and song lyrics.

The input of the classification model I build will be a numerical representation of the song's lyrics in the dataset, and the output will be class predictions for both artist and genre which will be evaluated using evaluation metrics like f1-score from the Scikit-learn module[1].

The input of the recommendations system will be an average word embedding of the lyric data in a song generated from Gensim's Word2Vec module[2], and the output will be the most similar songs to the specified song.

To accomplish these objectives, I will use models, a vectorizer, and evaluation metrics from sk-learn[1], NLTK's word tokenizer[3], Word2Vec embeddings from the Gensin library[1], and data upsampling from Imblearn[4].

4 Methodology

4.1 Data Pre-Processing

In order to be able to analyze the data and use it for building the models, it was necessary to clean the dataset since there were a lot of incorrect data points as a result of the Kaggle dataset being scraped from genius.com, which includes a lot of user uploaded songs. To start, the dataset had ~5 million songs. In order to narrow these down to a usable sample, I decided to first remove any songs tagged as the genre misc. or country, since neither contained enough quality examples to use in training, and misc. contained non-song data. After that I removed songs with features due to the risk of lyrics between artists overlapping, which would have impacted classification. I also removed any songs in languages other than English to focus on only English language data. Then, I narrowed the songs down by only selecting artists who had 20 songs with over 10,000 views in order to ensure every artist would have at least 20 songs to train from, and to ensure that the songs selected were actual songs and not unknown data that was uploaded to Genius by users. After preprocessing, the dataset contains ~44k songs and ~900 unique artists.

4.2 Tokenizing

The first step after preprocessing was to tokenize the lyrics since that is what I will use to create the input for both the classification and recommendation systems. To do this I used NLTK's word tokenizer[3] and python's re library to break the lyric data into tokens with any non-lyric data such as [intro] removed.

4.3 Baseline Model

After tokenizing the lyrics, I split the dataset into an 80/20 train/test split with a random_state=1 for consistency using sk-learn's train_test_split[1], and then started on a baseline model to compare my later classification models with. To do this, I used sk-learn's count vectorizer[1] to create a bag of words vectorization of the lyrics for each song, fit transformed on the training lyrics and transformed on the test lyrics to ensure no data bleeding

[1] https://scikit-learn.org/

[2] https://radimrehurek.com/gensim/models/word2vec.html

[3] https://www.nltk.org/api/nltk.tokenize.html

[4] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html

occurred between the train and test data. The baseline model I tested using this vectorization was the Naïve Bayes model from sk-learn's library[1].

4.4 Word2Vec Embeddings

After training the baseline model with a bag of words, I moved on to building the other classification models which will use Word2Vec embeddings trained on the tokenized lyrics. I decided to use Word2Vec embeddings due to their ability to capture word/token semantics which I expect will be more useful than a basic bag of words approach since they can more accurately represent the meaning/topic of a specific song. After training Gensim's Word2Vec embeddings[2] on all the training tokenized lyrics and applying it to the train and test lyric data, I had a 100 dimension representation of each token, and further analysis showed that these representations captured meaning well; for example the tokens most similar to "ice" (often used as slang for diamonds) were different words for watches and diamond types, rather than actual frozen water.

After checking that these token embeddings contained meaning related to the training data, I took the average of all tokens in a song, for each of the 43k songs, in order to have an average embedding of each song which I expect to act as representations of the song's topic.

4.5 Data Balancing

Finally, before working on the models I decided to upsample my data in an additional training dataset in order to test if by balancing the genres, I could improve the result of the models. I did this using imblearn's RandomOverSampler[4] which randomly samples and adds datapoints from minority classes to balance them with the majority. This results in an equal distribution of genres in the training data.

4.6 Song Recommendations

After obtaining the average song embeddings in *4.4 Word2Vec Embeddings* above, I was able to create a function which generates a specified number of song recommendations when given a song and artist. It does so by calculating Euclidian distance[5] between the specified song embedding and every other song embedding. This results in a list of the most "similar" or closest songs which can be sorted by distance and returned based on how many song recommendations were requested. The attatched Jupyter notebook shows examples of these recommendations for popular songs in each genre.

4.7 Classification Models

4.7.1 K-Nearest Neighbors

When starting building classification models, I first tried to consider how the data would be distributed. Initially, I had thought that similar songs would be close together and the most similar songs would be from the same artists and genres, which was why I started with a K-Nearest Neighbors model from scikit-learn[1], which calculates distance to the k nearest points and assigns a class based on the majority. I tested a number of n_neighbors to find which resulted in the best classification, and found that n_neighbors of 2 for artists and 10 for genre worked the best, due to there being so many more examples of each genre than each artist. After using the non-balanced data, I trained the same models on the upsampled data for genres.

4.7.2 Multi-layer Perceptron

After K-Nearest Neighbors, I moved on to a multilayer perceptron from scikit-learn[1] using the relu activation function, which I initially expected to perform fairly well on both genre and artist due to its ability to classify non-linear data and fit multiple activation functions which allow for this non-linearity. After testing a number of hidden layer sizes, I settled on 100, since this balanced results and computation time. The model uses early stopping to ensure that overfitting on the training data does not occur, and a maximum of 10 iterations due to an extremely long convergence time otherwise. I also tested this model with the balanced genre training data to see if performance would be improved.

4.7.3 Random Forest

Next, I implemented a RandomForest model from sk-learn[1] using 100 decision trees as the estimator, but only applied it to genre classification due to the computation time being extremely high when classifying artists since there are ~900 classes as opposed to only 4 genres. I had initially expected a

[5] https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

decent result from RandomForest, since I thought the ability to separate data into quadrants based on class would be useful in identifying similar songs.

4.7.3 Logistic Regression

Finally, the last model I tried was a Logistic Regression model from scikit-learn[1], which I had not initially expected to perform well due to its use mainly for binary classification and with data which has linear relationships. For the Logistic Regression model, I used a maximum iterations of 200 due to prevent the model from overfitting and to reduce the computation time.

5 Results

The system I built to answer my objectives in the Introduction includes both the Recommendation function, as well as the Classification Model to predict genre and artist from a song's lyric data.

5.1 Song Recommendation Function Results

One of the results of this system is a recommendation function which is capable of generating diverse recommendations when given a specific song and artist, and although the recommended songs may not match the exact vibe or style of the song due to only lyric data being used rather than audio, The song's topic and genre will generally be similar. In addition, I believe the recommendation function achieves my objective of generating recommendations in a different way from how Spotify does, and being able to generate recommendations without other user data, only lyric data.

5.2 Artist and Genre Classification Results

5.2.1 Baseline model

To predict artist and genre from given lyric data, the initial baseline Naïve Bayes with bag of words model I built performed surprisingly well for classifying genre, with an overall accuracy of .71, which is much better than random chance. However, one thing it struggles with is classifying r&b correctly, which is a trend throughout the models. This is likely due to a lack of datapoints for r&b compared to the other genres, and even upsampling the data does little to fix this issue. This struggle is shown with r&b having an f1-score of only .34 when compared to the other genres which are all above .50.

For classifying artist, the baseline model performed very poorly, with an accuracy of only around .04. f1-score was difficult to visualize due to the number of artists, but tended to be higher for artists with more songs in the dataset than the minimum of 20 songs set in *4.1 Data Preprocessing*. This is likely due to the number of classes to predict, with artists often overlapping in topics.

5.2.2 Classification Models with Word2Vec

My non-baseline models consisted of a Word2Vec representation of each song's lyrics as described in *4.4 Word2Vec Embeddings*. I had initially expected these models to perform much better in comparison to the baseline, due to Word2Vec's ability to capture a token's semantics. This hypothesis was confirmed when testing out a number of models using the Word2Vec embeddings.

The K-nearest Neighbors model, which I had initially expected to perform the best, had an artist accuracy of ~.07, and a genre accuracy of .64, with poor f1-scores for all genres besides rap and rock. For genre classification, this was a bit surprising since I had initially expected an improvement over the baseline model due to k-nearest neighbors' ability to classify based on close-by data points, which I had expected songs with the same genres to be. Even when using the balanced genre data to train the model, which boosted the f1 score of r&b by .08, the other classes suffered as a result. One explanation for this may be that the song embeddings struggle to accurately capture context, and a Bert embedding would be better for this purpose.

The results of the Multi-layer Perceptron saw a significant improvement over the KNN model for genre, with the only struggle being in predicting r&b, which had an f1 of .19. Aside from that, the other classes all had relatively high f1with an overall genre accuracy of .72. For artists as well, there was an improvement at an accuracy of ~.1. Again, when using the balanced genre data, there was a slight improvement in classifying r&b, but the other classes suffered as a result. The results for the RandomForest model were very similar to the results of the Multi-layer Perceptron, likely also due to the lack of contextualized embeddings and not enough data for each genre.

Finally, the Logistic Regression model saw the best performance with an artist accuracy of ~.16, and genre accuracy extremely similar to the Multi-layer Perceptron. One downside is the long computation time due to the time of converging with so many classes. I was somewhat surprised with these results since I did not expect the data to have much of a linear relationship, and instead thought it would be clustered in groups of datapoints with similar genres and artists, which is why I initially used K-Nearest Neighbors.

5.2.3 Classification Models Conclusion

In terms of artist classification, I had initially expected extremely poor performance due to the number of artist classes to predict, so I was not too surprised when the models struggled to predict artist with high accuracy. The genre classification performed better than expected, and all genres besides r&b were consistently predicted fairly accurately. I think overall, the models using Word2Vec showed a significant improvement over the Baseline model, and performed much better than random chance. For more information on the specific results of each individual model, the attached Jupyter Notebook contains a classification report for each along with further explanations and analysis of the results.

6 References

Spotify. 2024. Understanding Recommendations. *Spotify Safety and Privacy.*

Agrawal, Raghav. 2022. Music Genre Classification Project Using Machine Learning Techniques. *Analytics Vidhya.*

CarlosGDCJ. 2022. Genius Song Lyrics with Language Information. *Kaggle Datasets.*