Classifying a Song's Genre Using Spotify Metadata

Jeet Ajmani	**Jagjeet Singh**	**Kenil Shah**	**Ella Kuypers**
University of Victoria	University of Victoria	University of Victoria	University of Victoria
V00942302	V00970743	V00903842	V00910928

Introduction

For our project, our team has built a music genre classifier that uses only song metadata as input. Music metadata consists of features such as the song name, artist, country of origin, tempo, beats-per-minute, and much more. Using only the song metadata avoids using computationally intensive algorithms, which allows our genre classifier to run in real-time alongside high-intensity applications, like video games. As multiple members of the team have experience in music retrieval techniques using audio samples, we are well aware of the computations involved in doing genre classification using audio samples. These usually involve Fourier Transforms that cannot be performed in real-time in time-sensitive applications.

It is important to note that music metadata often includes the genre as a feature already. Nevertheless, we believe our project is still useful as our classifier can be used as a base to generate various features of a song given some metadata. For example, we could create a song name generator using AI that will label the music based on features like tempo, danceability, or genre. Given the smaller team size, we have wisely opted to narrow down the scope of our project to simply classify the genre. In the future, we could expand our project to perform tasks as described earlier, such as song name generation. Furthermore, using only the music metadata to classify songs could result in interesting clusterings that might not have been visible using traditional audio sampling.

Problem

As our goal is to perform genre classification using only the music metadata, the problem involves figuring out which features of the metadata are helpful in genre classification, and which features result in ambiguity. Furthermore, we require a large collection of accurately labelled songs to build a reasonable classifier. Finally, to successfully label genre, we require a dataset that has genre as a feature.

Background and Related Work

Music genre classification is not a new concept and there exist many genre classifiers that use the audio signal as an input to perform genre classification by performing beat and pitch detection. As mentioned earlier, this is a computationally expensive task that is difficult to perform within small time frames for applications that are time-sensitive. Using the song metadata is not a common

input vector as this involves already having processed a piece of music beforehand to obtain features like beats-per-minute, tempo, danceability, and more. However, song metadata is becoming more and more available due in part to companies like Spotify that maintain large databases of song information including very specific features like “speechiness” or “acousticness.” Spotify is a music app that uses many machine learning methods on its platform, thus, they have generated many feature vectors for that task. Our angle is to use this readily available metadata to see if it is possible to classify genres using only the information from this metadata. As mentioned before, the future expansion of this experiment is to attempt to use machine learning to generate labels given fabricated metadata.

Approach

The task of genre classification requires a reasonably large pool of song information that machine learning classifiers can build upon. The song information needs to include genre as a label as well in order for us to measure the success of our classifiers. Furthermore, we needed enough features in the dataset to create a reasonable classifier for genres. Thus, our first goal was to search for such a database. Fortunately, our team was able to obtain a dataset that includes all the necessary information for our classifier. Our approach to classification was as follows: First, preprocess the dataset into a usable form that can be fed into our classifiers; next, compare the base performance of multiple classifiers to choose which model would work the best; subsequently, based on these performance metrics, apply feature selection methods to increase the performance of said classifiers and gain insight as to which exact features are most relevant to the task of genre classification; lastly, perform hyperparameter tuning on the best performing machine learning model.

Dataset

The project requires a dataset containing metadata about songs rather than audio file data. The dataset chosen for this project is a Spotify song metadata database containing metadata from 3100 songs extracted from Spotify’s Daily Top 200 charts over three years (2017-2020) [1]. The dataset was found on the website Kaggle and created by six university students for a Big Data class. We chose the dataset because it contains many data entries, and each song item has lots of features. Each song has a parent genre field that served as the target label for our project. The range of values for the parent genre are hip hop, pop, dance/electronic, rock, r&b/soul, boy band, k-pop, indie, country, Latin, funk, & else. Each song in the database has 51 metadata features, including artist, country, explicit, danceability, and loudness. There are two notable downsides to using the database: the disproportionate number of songs from particular genres and the lack of cleaning. The data comes from the top 200 charts, so nearly half of the songs are from the pop and hip hop genres. As for the lack of cleaning, some rows in the original dataset have null values in key fields.

Initial Experiment

The initial experiment was conducted to gain familiarity with processing the dataset and to determine the classifier model the project would use. Initial tests were written in Python on Google Colab notebooks using the machine learning library Sci-kit Learn. To reduce the number of features for the tests, the features consisted of Title, Artist, danceability, energy, loudness, liveliness, valence, tempo, duration (in ms), acoustics, and explicitness (1 or 0). These features

were chosen by team members based on presumed relevancy to the classification problem without the use of feature selection methods. A label encoder was applied to all of the chosen features, simply to convert categorical data into numerical data. Experiments were separated based on the preprocessing method used on the dataset. Three preprocessing options were explored: no numerical preprocessing, min-max preprocessing, and standard scaler preprocessing. Table 1 displays the training and test error results.

Table 1: Training & test errors of classification methods from initial experiments

	Experiment 1: *No Numerical Preprocessing*		**Experiment 2:** *Minmax Preprocessing*		**Experiment 3:** *Standard Scaler Preprocessing*	
Algorithm Name	**Training Error**	**Test Error**	**Training Error**	**Test Error**	**Training Error**	**Test Error**
Decision Tree	0.0	0.337	0.0	0.315	0.0	0.335
Random Forest	0.139	0.397	0.0	0.354	0.0	0.364
K Nearest Neighbors	0.445	0.631	0.394	0.535	0.388	0.561
Linear SVM	Did not converge	Did not converge	0.512	0.521	0.516	0.530
Logistic Regression (sag)	0.582	0.402	0.505	0.513	0.502	0.519
Logistic Regression (lbfgs)	Did not converge	Did not converge	0.505	0.513	0.502	0.519

Preliminary results were not promising for any of the algorithms besides the Decision Tree and Random Forest methods. Additionally, the drastically low training errors for the tree-based classification algorithms imply overfitting. An additional test was conducted with the decision trees to attempt a binary classification of hip-hop songs vs. metal songs to observe how the feature set performed in a binary classification scenario. A decision tree classifier was able to classify the problem with a 3.8% test error and near 0% train error, which is similar to the multi-class and again implied overfitting.

Based on the initial experimentation results, the team concluded that more preprocessing had to be done on the dataset to improve the results. A classifier model was not selected yet, because the error results are too similar for meaningful comparison. Also, the project moved forward with standard scaling as the scaling method of choice. Besides preprocessing, the focus shifted to experimentation with feature selection and hyper-parameter tuning to optimize the classification algorithms.

Experimental Results

Following the initial trivial experiments using the raw dataset with minimal numerical preprocessing, we conducted two further experiments to determine the optimal machine learning model for our problem and tune and apply the selected model to try and achieve a relatively low test error.

Experiment 1: Determining the Optimal Machine Learning Model

Firstly, both the feature matrix and the target vector required intense preprocessing. Due to Scikit-Learn's classification models restricting all features to only numerical values, to utilize a non-numerical feature such as a song's artist, we applied one-hot encoding to signify the relationship between song and artist. Our specific usage of one-hot encoding created a new column for each unique artist and assigned a 1 to the rows corresponding to a song made by a particular artist, with all other rows set to 0. This process resulted in more than 1400 new columns (features). Still, it successfully converted a nominal (non-numerical) feature to a numerical feature in a way that is recognizable by Scikit-Learn's classifiers. Additionally, the dataset contained features extracted from the lyrics of only English-language songs. If a song was in another language, the row had missing values. We experimented with dropping all rows containing missing values, initializing the missing values to 0, and ignoring the features with missing values. As a final measure, we standardized all numerical features using Scikit-Learn's StandardScaler, which centered the data and allowed our Linear SVM model to converge. For the target vector, Spotify's genre label was often too hyper-specific (UK Hip Hop, Dance Pop, Canadian Contemporary R&B). This column contained 256 possible classes, some of which only included a single song. Instead, we decided to use a different column from the dataset, which contained 18 'parent' genres (Hip Hop, Pop, R&B) and included plenty of songs in each class. To narrow down the number of target labels even further, we grouped the genres 'Rap' and 'Trap' with 'Hip Hop,' 'House' with 'Dance/Electronic,' and 'Metal' with 'Rock.' Therefore, our target vector for the first experiment contained 14 total classes (genres).

Now we were left with 3100 entries in our dataset with a feature vector dimension of 1491, out of which 51 were numeric features and the rest were the result of one-hot encoding. To improve the accuracy of our model and reduce the dimensions of our features we employed the following feature selection methods:

1) Tree Based Feature Selection
2) Recursive Feature Elimination
3) ANOVA F-test

Before we began, we decided to determine a baseline on the entire feature set. To combat the problem of overfitting we used 5-fold stratified cross validation. The baseline results in table 2 showed that SVM with linear kernel performed significantly better than the rest of the classifiers.

Table 2: Performance of models without feature elimination

	Decision Trees	Random Forests	K-nearest neighbours	Logistic Regression	Linear SVM
Train Error	0.333	0.294	0.415	0.316	0.215
Test Error	0.345	0.301	0.442	0.298	0.192

For Tree-Based Feature Selection, we used a random forest classifier to assign importance scores to all our features. Then using the mean of these scores we discarded all features whose importance score was below the mean, leaving us with a feature vector dimension of 134. As observed in table 3, random forests showed a slight improvement, although to the expense of the remaining classifiers.

Table 3: Performance of models with tree-based feature selection

	Decision Trees	Random Forests	K-nearest neighbors	Logistic Regression	Linear SVM
Train Error	0.358	0.282	0.419	0.322	0.234
Test Error	0.371	0.279	0.442	0.305	0.213

Next was Recursive Feature Elimination (RFE). The idea behind RFE is to use a model to assign importance scores to our features. For this part, we decided to use linear SVM as our model, as our baseline experiments without feature selection gave the best results for SVM and we hoped that using SVM here would improve those results. Then, we removed the least important feature, fit the model, and calculated the training error using cross-validation. Next, we removed the second least important feature and repeated the process again. We continued with this process until no features remained. The feature set which minimized the training error was chosen as the optimal feature set. This left us with a feature vector dimension of 1021. The numeric features that were selected were - explicit, danceability, energy, loudness, speechiness, valence. Table 4 shows the significant improvement of tree-based classifiers, but surprisingly no improvement for Linear SVM.

Table 4: Performance of models with Recursive Feature Elimination (RFE)

	Decision Trees	Random Forests	K-nearest neighbors	Logistic Regression	Linear SVM
Train Error	0.289	0.219	0.321	0.318	0.215
Test Error	0.263	0.203	0.313	0.322	0.204

Last was the ANOVA F-Test. ANOVA uses a statistical test called the F-test that calculates the independence of features from the target variables. This is extremely useful when input variables are numeric and target variables are categorical. After assigning F-scores to the features, we recursively add one feature at a time in order of descending F-score, fit the model, and calculate the training error using cross-validation. We continue doing this until all features are used. The feature set which minimizes the training error is chosen as the optimal feature set. This left us with a feature vector dimension of 851. All 51 numeric features were selected. As seen in Table 5, this method showed small improvements for KNN, logistic regression and linear SVM, but showed minor to no improvement for tree-based classifiers.

Table 5: Performance of models with Anova F-Test

	Decision Trees	Random Forests	K-nearest neighbors	Logistic Regression	Linear SVM
Train Error	0.338	0.283	0.391	0.324	0.224
Test Error	0.282	0.234	0.335	0.266	0.152

After analyzing the results of the whole experiment, we decided to proceed with a Linear SVM as our preferred model and delved deeper to try and further improve its performance.

Experiment 2: Tuning and Applying the Previously Selected Model

Despite decent results from the first experiment, we decided to try applying even more preprocessing. Specifically, if multiple artists made a song together, the dataset treated them as a single artist. For example, suppose that "ArtistA," "ArtistB," and "ArtistC" made "Song1" together. The dataset considered the sole artist for "Song1" to be a string containing the three artists' names separated by dashes ("Artist1 - Artist2 - Artist3"). Having a single agglomerated group of artists representing a song's artist was slightly problematic because of the possibility that some of those artists inside the group may have made other songs, whereas the group may have only made one. For this reason, we decided to split the artists of each song into lists, with a minimum of 1 artist and a maximum of 13 artists (specifically the song "Don't Shoot" by The Game, featuring Rick Ross, 2 Chainz, Diddy, Fabolous, Wale, DJ Khaled, Swizz Beatz, Yo Gotti, Curren$y, Problem, King Pharaoh, and TGT). To further emphasize why this was an important decision, each artist in this group had other songs throughout the dataset, but "Don't Shoot" was the only song that included all 13 of them together. After creating a list of artists for each song, we formed a set containing every artist exactly once and encoded a 1 if an artist was present on a specified song, or a 0 otherwise. This method led to about 900 extra columns, whereas the previous method resulted in over 1400. Additionally, we discarded the song title column due to its irrelevance in classifying a song's genre, and dropped all rows with missing values, resulting in fewer songs in each genre class. At this point, since the genres 'Reggaeton' and 'Bolero' contained fewer than 5 songs each, we grouped them with the label 'Else,' which left us with a total of 12 classes in our target vector. Figure 1 shows the distribution of genres in the final preprocessed dataset used in Experiment 2.

As we decided to proceed with a Linear SVM as our chosen classifier, we used cross-validation to tune the regularization parameter C between 10^{-8} and 10^{6}. We observed that the value of $C = 1.0$ resulted in the lowest test error. Figure 2 shows the result of the hyperparameter tuning. The other properties of our model were l_2 regularization and squared hinge loss.

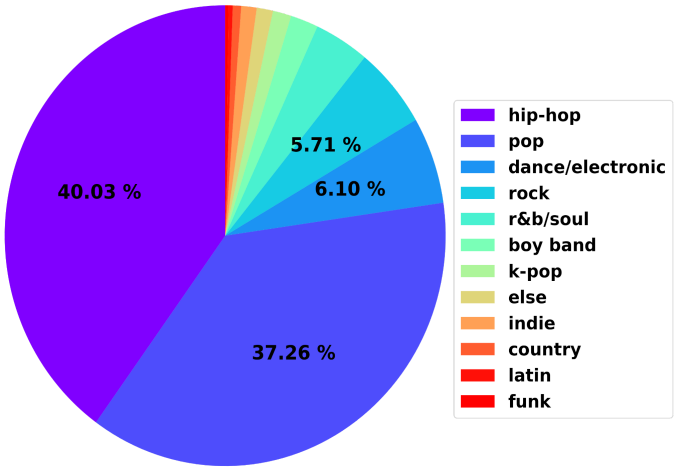

Figure 1: Pie chart outlining the distribution of genres for Experiment 2.

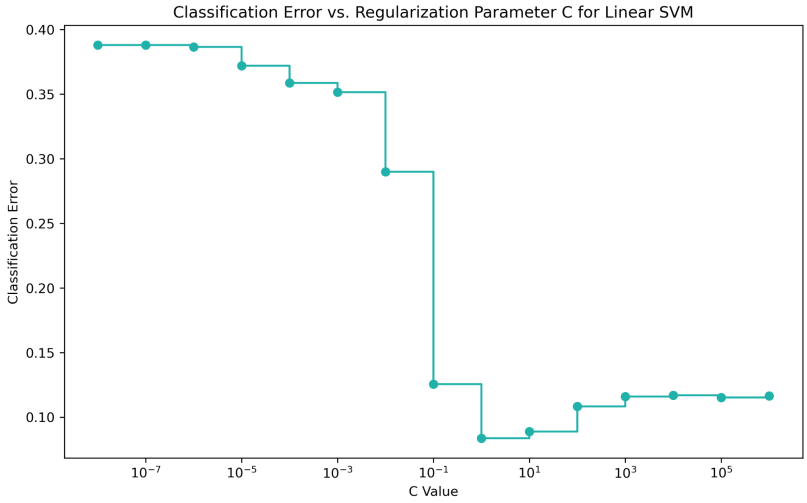

Figure 2: Graph of classification error vs. regularization parameter C. Lowest error at 10^{0} (or 1).

Conclusion

The team successfully built an effective multi-class model for classifying the genre of songs using metadata. The final Linear SVM classification model built for the project classifies song metadata with a test error of 7.55%. The test error of 7.55% surpasses our initial goal of achieving an error

of at most 15%. Figure 3 displays the improvement of the test error over the project's many iterations.

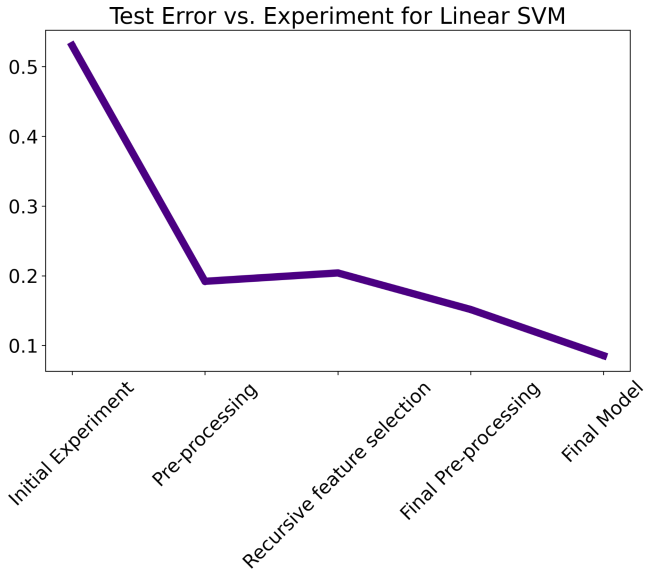

Figure 3: Improvement in test error over each project milestone

The sharpest drop in test error came from a boost in cleaning the dataset after the initial experiments. Removing rows containing missing values and replacing grouped artists with binary artist features improved the error significantly from the initial experiments' results. Hyperparameter tuning and recursive feature selection optimized the linear SVM model to achieve the final test error of 7.55%.

With more time to develop the project, the team would have tested the model on an additional dataset. Results from another database would help refine the model and further validate its effectiveness of the model. Although the database used to build the project's final model was large, the disproportionate numbers of songs from particular genres leave room for improvement. Other avenues of expansion considered by the team included broadening into genre classification for non-English music. Non-English songs from the database are excluded from the model because many are missing values, including the genre label.

References

[1] "Spotify HUGE database - daily charts over 3 years | Kaggle." Accessed April 15, 2022. https://www.kaggle.com/pepepython/spotify-huge-database-daily-charts-over-3-years.

[2] "Scikit-learn: Machine Learning in Python", Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[3] "Recursive Feature Elimination (RFE) for Feature Selection in Python" https://machinelearningmastery.com/rfe-feature-selection-in-python/

Appendix A: Graphs

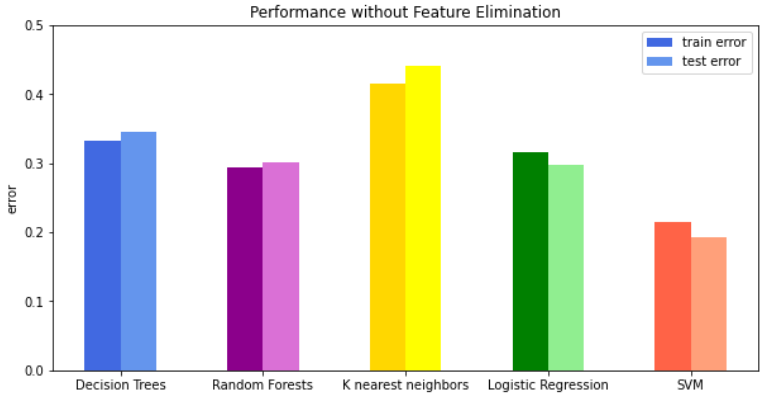

Figure 1: Baseline performance of classifiers without feature selection

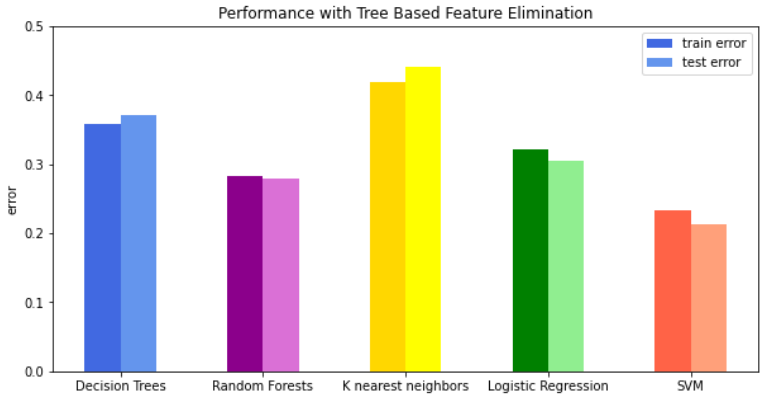

Figure 2: Performance of classifiers with tree-based feature selection

Figure 3: Performance of classifiers with recursive feature elimination

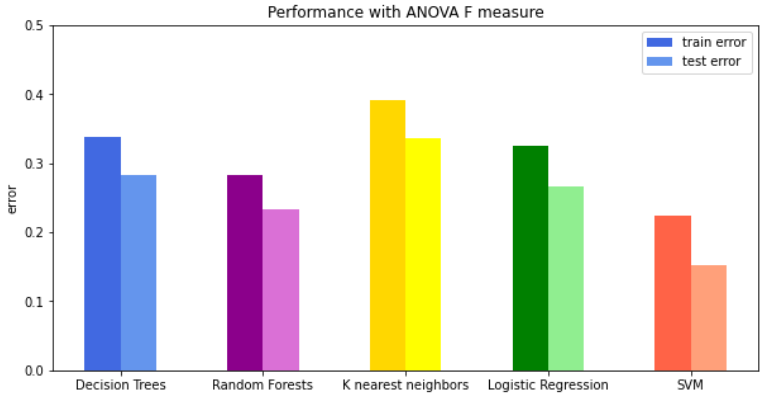

Figure 4: Performance of classifiers with feature selection using ANOVA F-test