Classification of Songs Based on Lyric Frequent Pattern Mining

Vincent Farrell
Faculty of Science
University of Manitoba
Winnipeg, Canada
farrelvs@myumanitoba.ca

Emma DePape
Faculty of Science
University of Manitoba
Winnipeg, Canada
depapee@myumanitoba.ca

Akashkumar Ghelani
Faculty of Science
University of Manitoba
Winnpeg, Canada
ghelanas@myumanitoba.ca

Roselle Deinla
Faculty of Science
University of Manitoba
Winnipeg, Canada
deinlar@myumanitoba.ca

***Abstract*— As the music industry grows increasingly lucrative and more non-labelled artists emerge, the existence of a reliable genre classifier would provide a powerful tool. Genres themselves are structured yet borderless categories. By cracking the code of a genre, an artist can find grounds for potential success. Using mlxtend's Apriori-based Frequent Pattern Mining algorithm and Class Association Rules Generator, a subset of the 'Million Songs' database is mined for lyric-based Association Rules of high confidence. A Classification Based on an Association model is then constructed around the extracted lyrics to predict the genre of inputs derived from a song.**

I. INTRODUCTION

The artistry behind good lyricism is no easy feat. Subtle intricacies and intonations in a song can distinguish between what is considered a "timeless classic" and what will fade into obscurity. While authenticity cannot always be bundled with an artificially generated piece, a sufficient foundation and means of comparison can provide a good evaluation of a song's writing. Not all music contains words, nor must it have any hidden meaning. Yet identifying patterns in those that do, representing the majority of popular music, could prove a worthwhile analysis given how vast and accessible songs and their lyrics are.

Our goal is to aid artists and songwriters in their creative process by allowing a way to check their drafts to see if it aligns with their target genre. Record labels may want to transition their artist(s) into a new field, whether to keep up with current trends or find a sound that better suits their talents. With the influence of public streaming platforms (such as Spotify and Soundcloud) and social media, publishing and distributing music is much easier without the resources a label would provide. Given the sea of rival creatives, it can be challenging for such individuals to recognize their names and listen to their music. Allowing small artists insight into their works can give them the edge to stand out and provide direction without a team. Perhaps one wishes to emulate a particular artist. Analyzing their lyrics could produce a similar piece or tribute to their likeliness. In addition, intimidation from experimenting and branching out to a different genre can be mitigated.

A piece could be constructed by analyzing which frequent patterns (FPs) are commonly seen in the target and incorporated accordingly. The classification itself provides a way of measuring current writing direction. FP findings can bring less arbitrariness to the overlapping nature of genres and give insight into what sequence of words produce a "hit" track.

Musical genre is also widely used to classify and describe titles by the music industry and the consumers. Classifying genre is inherently a difficult task because features often overlap between different genres. Also, the key features needed to declare a song a certain genre are not officially defined. A lot of work has been done on classifying songs based on their timber, bass, and other audio qualities. However, there has been a lack of work studying the impact of lyrics on the classification of songs by genre.

The dataset we have chosen to mine is the "Million Songs Dataset" (MSD) [1], compiled by individuals at LabROSA and The Echo Nest. It consists of over a million songs spread over ten genres and tracks the counts of unique words per song, among other features. Due to copyright issues, the actual sentences are not provided. Instead, a count of all words that occur per song are given to sneak around the issue.

The algorithm chosen to mine the FPs comes from the mlxtend [10] python package. The later classification models will be derived using functions from the scikit-learn (SK Learn) [11] python package.

II. BACKGROUND & RELATED WORK

A. Genres

A genre is the simplest form of categorizing and describing a song. A genre label aims to encapsulate overall feelings and emotions one experiences upon listening. As described by Aucouturier and Pachet [15], "genres are intrinsically ill-defined". Interpretation will significantly differ among those musically inclined to the everyday listener. A genre is not a strict tag; many sub-categories and blended styles exist. In reality, it is not entirely correct to prescribe a label to a song from lyrics alone. However, listeners with tons of song exposure can pick out and expect certain lines and patterns. The "main" genres found in our dataset are good umbrella terms and catchalls for most lyricized songs.

Classes:

- Punk
- Pop
- Country
- Electronic
- Rock
- Metal
- Blues
- Soul
- Folk
- Jazz

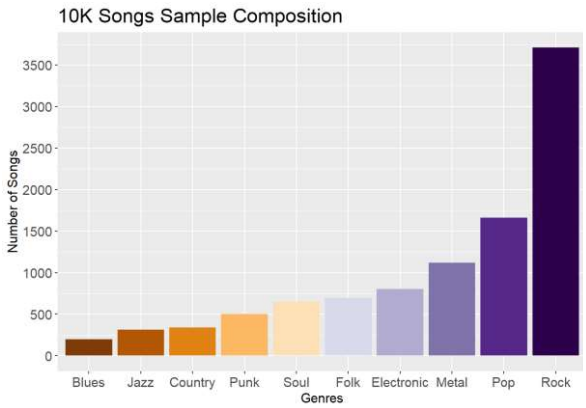

Fig. 1. Genre Breakdown of the subset of the MSD

B. *Machine Learning Models*

Machine Learning (ML) is "the technique that improves system performance by learning from experience via computational methods" [7]. Supervised Learning is the most suitable branch of ML for our purposes. Supervised Learning aims to predict a target from a given set of features. Notably, the training data fed into the supervised learning algorithm includes the desired solutions. More specifically, we want to perform a supervised learning task called classification. Classification aims to categorize a new observation into a specific class after being trained on already classified examples. The target we wish to predict is a song's genre, which will be labelled from the ten respective genres accordingly based on FP lyrics as features. Several ML models are designed to handle such labelling, and the models listed below have resulted in the highest evaluation metrics.

1) Stochastic Gradient Descent (SGD)

Gradient Descent (GD) is a method that adjusts a model's parameters iteratively to minimize some cost function [6]. This cost function is generally defined as the error between the hypothesized value of an observation [] and its actual value. The gradient of the cost function is calculated (the amount of times depends on the variation), and parameters are positioned to face what would bring the cost function to a minimum (in the opposite of the gradient). SGD incorporates its stochastic quality by selecting random instances during training and calculating the gradient for those instances. This process of adjusting the parameters is shown below in Figure 2. Final parameter values are not the most optimized, but this algorithm is more computationally efficient.

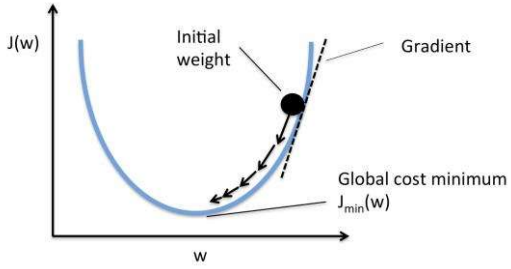

Fig. 2. Image explaining the SGD process.

2) Gaussian Naïve Bayes

Bayes' Rule for two events, A and B is defined as:

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (1)$$

Which calculates the "flipped" conditional probability given the known marginal probabilities and conditional probability of A given B. In machine learning, Baye's rule is applied where "C" represents either "true" or "false" for a single class, and "X" represents a feature [7].

$$P(C_{true}|X) = \frac{P(X|C_{true})P(C)}{P(X)} \quad (2)$$

$$P(C_{false}|X) = \frac{P(X|C_{false})P(C^c)}{P(X)} \quad (3)$$

It is typically easier to compute an estimate for the class-conditioned probability (feature given a class) and the probability of being confirmed for the class compared to estimating their joint probability (hence Baye's definition over the standard conditional probability). When calculating the probability for a class conditioned upon multiple features, estimates become computationally more challenging. To combat this, a Gaussian Naïve Bayes classifier assumes independence on the influence of each feature on prediction results and that each class follows a Gaussian distribution. Instead of estimating the joint probability of all features conditioned, we can calculate the different class-conditional probabilities and multiply them per the definition of event independence. The probability that a class is positive or negative given a set of features is then acquired by evaluating and comparing the below probabilities:

$$P(C_{true}|X_i) = \frac{P(X_1|C_{true})\cdots P(X_n|C_{true})P(C)}{P(X_1)\cdots P(X_n)} \quad (4)$$

$$P(C_{false}|X_i) = \frac{P(X_1|C_{false})\cdots P(X_n|C_{false})P(C^c)}{P(X_1)\cdots P(X_n)} \quad (5)$$

Whichever probability is higher marks the predicted class of the target. For each data point, the z-score distance between that point and each class-mean is calculated, namely the distance from the class mean divided by the standard deviation of that class, this is shown below in Figure 3.

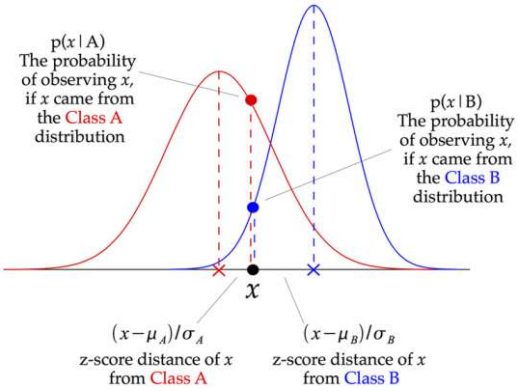

Fig. 3. Illustration of how a Gaussian Naïve Bayes classifier works.

3) Support Vector Machines (SVM)

In a linear sense, given a training set of points $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, $y_i \in \{-1, +1\}$ represents the class that a point x_i in some Real-valued space belongs to [8]. The optimized classification of such points is to fit a hyperplane (a subset of the Real-valued space that almost satisfies subspace qualities) such that the distance from the nearest points in class $y_i = 1$, and those in $y_i = -1$ to the hyperplane are maximized. If it is possible for the points to be separated linearly, then two parallel hyperplanes can be fitted to divide the points by class. The area between the hyperplanes is known as the "margin", and a "maximum-margin" hyperplane is then fitted right in the center of the region. Support vectors are such points that lie closest to the maximum-margin and determine its placement.

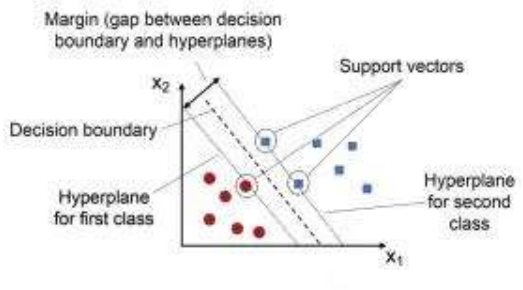

Fig. 4. Illustrative example of SVC.

Through SK-Learn, the original binary-class method of an SVM model can be used for multi-class identification by creating n_classes * (n_classes - 1) / 2 classifiers [11]. The results of these classifiers are then "monotonically transform(ed) to a 'one-vs-rest' decision function of shape".

4) Decision Trees (DT)

A Decision Tree is an ML model derived from a binary tree structure. DTs are non-parametric models and do not require any assumptions about the data. Given their nature, a DT can more easily handle categorical variables and convert them into binary questions to be stored in a node. Following down the tree path of "yes" or "no" (or any other binary outcome) will eventually result in hitting a leaf, which becomes the predicted class of the observation. The greater the "purity" of a leaf, the better the model. The purity of a leaf is determined by whether or not all observations reaching the leaf belong to one class. All observations in one class make a leaf pure. Observations that are contained between two classes are considered impure. The less disparity, the greater the impurity.

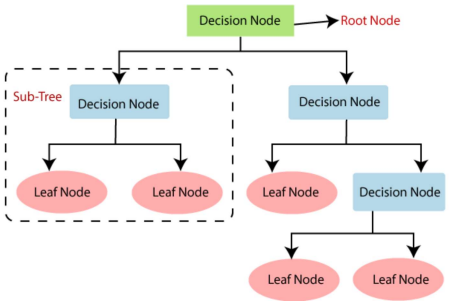

Fig 5. Diagram explaining a decision tree.

5) Random Forest (RF)

A Random Forest model is an amalgamation of DTs. An RF uses ensemble learning to combine less powerful predictors into one greater super predictor. The general algorithm of an RF is to bootstrap aggregate DTs together to increase tree variability. Then, a subset of random features is selected after each iteration to combat the correlation between trees and improve model performance. In theory, an RF model is designed to better handle multi-class classification and higher-dimensional data than a single decision tree. RFs, compared to DT, are more computationally complex.

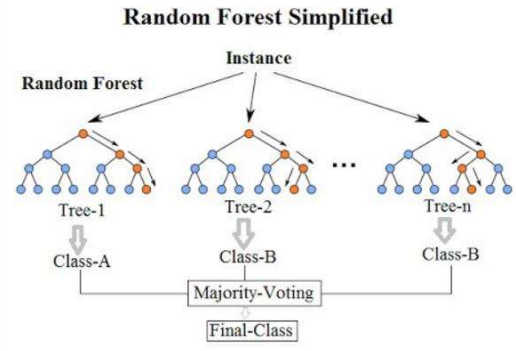

Fig. 6. Diagram illustrating a random forest.

6) Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a technique used to reduce the number of dimensions in a dataset while

preserving the maximum variance. PCA will reduce the number of principal components to those that include the most information available to be the primary features. Reducing the number of features using PCA may lead

7) Performance Metrics

Evaluating the performance of an ML model is one of the important steps while building an effective model. To evaluate the performance or quality of the model, different metrics are used, and these metrics are known as performance metrics [13]. The most commonly used metric is accuracy, which is determined as the number of correct predictions to the total number of predictions and is defined as:

$$Accuracy = \frac{number\ of\ correct\ predictions}{total\ number\ of\ predictions} \quad (6)$$

Accuracy is easily biased when used on skewed datasets. Since the song dataset does not have an equal number of songs for each genre, we need to choose a better metric that is not as affected by imbalance. Thus, we decided to use a metric called balanced accuracy. This is defined as the arithmetic mean of sensitivity and specificity and is used for imbalanced data. Sensitivity is the true positive rate and specificity is the true negative rate. The formula for balanced accuracy is defined as:

$$Balanced\ Accuracy = \frac{Sensitivity + Specifity}{2} \quad (7)$$

Other metrics that are commonly used for classification tasks are precision and recall. Precision determines the proportion of positive predictions that were actually correct. It is calculated using the number of true positives (TP) and false positives (FP). The formula for precision is defined as:

$$Precision = \frac{TP}{TP+FP} \quad (8)$$

Recall determines the proportion of actual positive predictions that were identified incorrectly. It is calculated using true positives and false negatives (FN). The formula for recall is defined as:

$$Recall = \frac{TP}{TP+FN} \quad (9)$$

A confusion matrix is a tabular representation of prediction outcomes of a classifier, which is used to describe the performance of the classification model on a set of test data when true values are known. A tabular representation of a binary confusion matrix can be seen in Figure 6.

The final metric we will be using to evaluate our models is the F1 score. The F1 score is described as the harmonic mean of both precision and recall, assigning equal weight to each of them. The formula is defined as:

$$F1 - score = 2 * \frac{precision * recall}{precision + recall} \quad (10)$$

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Fig. 7. A binary confusion matrix

Overall, an ML model can be evaluated using many different performance metrics. The best model is chosen in a way that finds a balance for all the metrics calculated, in the best model all metrics would be high.

8) Hyperparameter Tuning

Hyperparameters are higher-level parameters set manually before starting a model's training. They are based on properties such as the data's characteristics and the algorithm's capacity to learn [12]. An essential step in building an ML model is exploring these hyperparameters to find the best values that lead to the highest performance metrics. Hyperparameter tuning is the process of optimizing those values for the model. Grid search is one method that can be used to complete this optimization. Grid search is done by going through all possible combinations of the hyperparameter values [12]. This method is time-consuming, so it should only be used for a lower number of hyperparameters.

The hyperparameters needed to be tuned differently for each ML model. For SGD, the parameters that can be adjusted are the loss function, the regularization term or penalty, and the alpha value (this multiplies the regularization term and determines how strong the regularization is) [11]. For Gaussian Naïve Bayes, only a single hyperparameter can be tuned, which is "var_smoothing" in SK Learn or the portion of the most significant variance of all features added to variances for calculation stability. For a Decision Tree, the parameters that can be adjusted are the criterion (the function that measures the quality of the split), the strategy used to choose the split at each node, the maximum depth of the tree, and the number of features to consider when looking for the best split [11]. In a Random Forest model, the number of trees in the forest, the maximum depth of the tree, the criterion, and the number of features to consider when looking for the best split. Finally, for SVC, the hyperparameters are the regularization parameter, the kernel type, and the kernel coefficient. Therefore, in grid search, we create parameter grids with potential values for each of the mentioned hyperparameters, and the search will fit a model for each combination of hyperparameter values. The search returns the model with the highest metric value chosen to optimize.

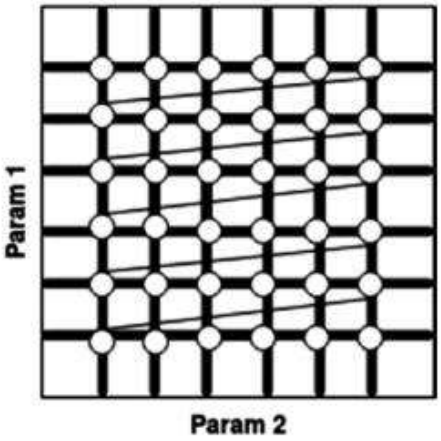

Fig. 8. A grid search going through each possible combination of two hyperparameters.

C. Related Works

Studies conducted by students at Stanford University and the University of Victoria both emphasize that most prior works have attempted to classify songs solely on their instrumental components and rhythmic structure. A study by Boonyanit and Dahl [4] aimed to see how effective lyric-based classification is and if it could be combined with non-verbal models to increase their accuracy. Their study did not involve data mining but instead used text embedding to identify word similarity. Word for word and, eventually, concatenating a song's lyrics were used as features in logistic regression and Long Short-Term Memory models.

Yang's study [5] more closely resembles ours. The data used in the study comes from the same Million Song Dataset we are derived from. A data-mining-based technique known as native language processing (NLP) is used to analyze the MSD for the top 20 words of each genre (removing any pronouns in the count). The words are stored as a feature called "Bag-of-Words", where a particular lyric is stored in a dictionary with each word in the lyric associated with a frequency. The "Bag-of-Words" are then used as features in ML models, and input songs are classified accordingly. Instead of filtering for the most popular or similar singletons, we want to mine frequent strings of words to use as features in different ML models.

III. Mining Process & Construction

In this section we will describe our machine learning model solution to genre classification.

A. Data Collection

The "Million Songs Dataset" is stored as separate CSV files for tables including different information. We were interested in the three tables containing the song collection, the lyrics, and the user-defined tags/genre. We exported the files as SQLite databases for ease of joining. We started by taking a random subset of 10,000 songs to make the size more manageable for analysis. Then we took the subset tables and combined them with a natural join into a single table, joining by trackID. The final step in data collection was selecting the specific columns we were interested in analyzing. Those columns were "genre", "trackID", "title", "year", and "lyrics". We then exported this final table into a single CSV file.

B. Pre-processing

The first step in pre-processing our data was dealing with the genre column. Since the genre/tags were user-defined, the column had multiple tags for each song. These tags also included terms that were not genres but other user descriptors. We dealt with this issue by selecting one genre for each song that matched one of the ten classes we planned to try to predict.

One step we took during pre-processing was removing non-alphabetic words from the dataset. We chose to remove them for simplicity and since our focus is on the words. The dataset was trimmed down from 889,971 rows to 886,382 rows, resulting in a difference of only 3589. Although the difference was insignificant, this step can improve the quality of the data and the performance of the subsequent data mining algorithms.

Another step we took in cleaning the dataset was removing common words in the English language. Some examples of the words we deemed typical were "I", "you", "they", "we", and others. We removed a select number of these words and words that only appeared for a single song out of the 10,000 songs. We made this decision in hopes of improving our future model's accuracy.

The following action we took to prepare our data for analysis dealt with the fact that each row contained only a single word and many rows for each song. We fixed this issue by combining the word column into a list of all the words for each song. In terms of data mining, we treated the songs as transactions. The next step in preparing the data for mining was encoding the genres into digits (0 to 9) using a label encoder from SK Learn. We then separated each genre into separate datasets, so we obtained ten different datasets. We decided to do this so that the issue of imbalance between genre classes could be. We wanted to avoid a single genre overtaking the support because one genre has a higher number of songs than another. Next, we inserted the encoded genre into the list of words for each song. We did this so that we could mine class association rules. Class association rules are association rules that only contain a single class in the consequent. That is what we plan to use for making predictions. To prepare the songs for mining using mlxtend's Apriori function, convert the song words into actual transactions using the TransactionEncoder function from mlxtend. This converts the word list into a vertical transaction vector. Our data is ready for frequent pattern mining, which we will discuss in the subsequent section.

C. Frequent Pattern Mining

Our plan for prediction is to use frequent patterns mined from the word lists to form class association rules. We originally planned to use the pyARC package to mine the frequent patterns and form the association rules. However, this package ended up being quite complicated to use. Therefore, we switched to the mlxtend package, which was much more flexible and user-friendly. We started by mining each genre separately using the Apriori algorithm to obtain frequent patterns associated with each genre. Once we obtained all the frequent patterns, we selected the ones with support more significant than 0.1 as our

final patterns for each genre. We then combined the ten frequent pattern datasets into one for all the genre's frequent patterns. From these frequent patterns, we selected the ones that contain genres, i.e., the frequent patterns containing digits. The following table contains some examples of frequent patterns, their support values, and the genre they belong to.

Table 1. Frequent Pattern Examples

genre	words	support	confidence
pop	['all']	0.58027	1
metal	['be', 'will']	0.37745	1
soul	['babi', 'all', 'love']	0.25038	1
folk	['come', 'so', 'all', 'will']	0.10571	1

Then using mlxtend's association_rules function, we generated association rules which we then selected only the ones where the consequent was a digit, the encoded genre. We noticed many duplicate antecedents for multiple different genres as a consequence. We dealt with these by only keeping the ones with the highest support. The following table contains some examples of the mined class association rules.

Table 2. Class Association Rules

Antecedent	Consequent
{'have', 'know', 'love'}	soul
{'eye'}	metal
{'when', 'heart'}	country
{'time', 'down', 'know'}	blues

We will use these antecedents as our predictor variable in our classification model. Next, we will explore some initial statistical analysis of our data and the frequent patterns to see if we can come to any preliminary conclusions.

D. Preliminary Statistical Analysis

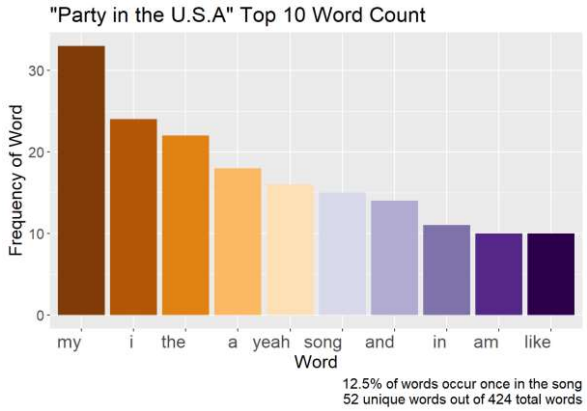

Fig. 9. Ten most frequent singletons found in "Party in the U.S.A"

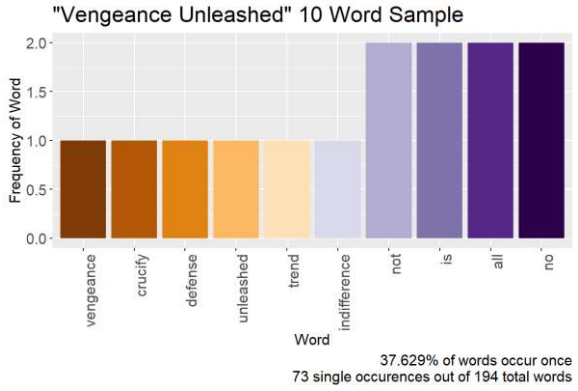

Fig. 10. Ten word singleton sample from "Vengeance Unleashed"

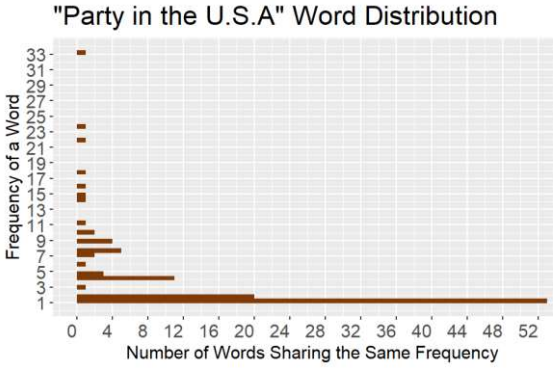

Fig. 11. Word Histogram for "Party in the U.S.A"

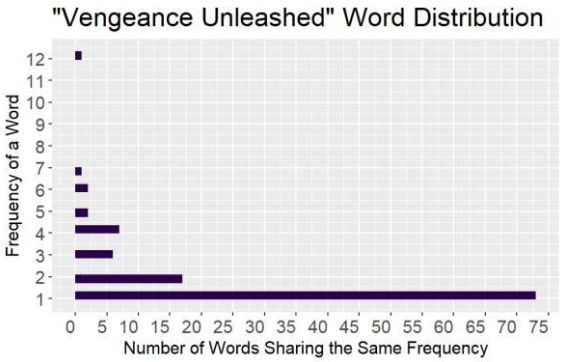

Fig 12. Word Histogram for "Vengeance Unleashed"

A look into a song's word distribution provides further reasoning on how plausible using lyrics as features in

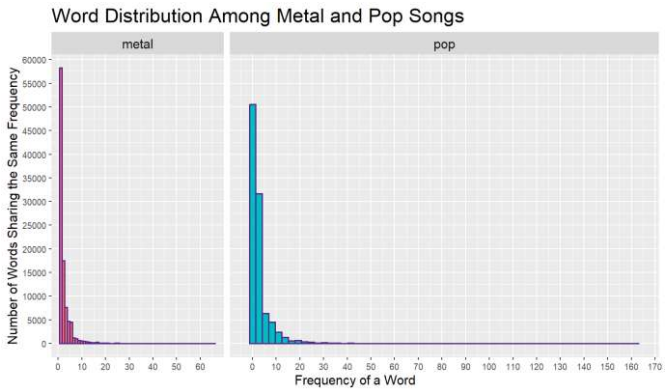

Fig. 13. Word Distribution between all Metal Songs and an Adjusted amount of Pop Songs

classification is. Fig. 9 shows the top ten most frequent words for the famous pop song "Party in the U.S.A". Unsurprisingly, the most frequent words are common pronouns and conjunctions. However, the count for such words is relatively high compared to the metal song "Vengeance Unleashed" found in Fig. 10. The most frequent word is "the", with a count of twelve, followed by "a", with a count of seven. Even when accounting that "Party in the U.S.A" has over twice as many words compared to "Vengeance Unleashed", there exists much more repetition in the pop song. A ten-word sample is taken from the metal song for a more exciting glimpse. Quite a few of the words are undoubtedly characteristic and not what most would expect to be found in a pop song, further supporting there are distinct enough differences between genres through diction.

For enhanced readability, flipped histograms of the total word distributions for both the songs have been created (Fig. 11 and Fig. 12). The y-axis represents how many times a specific word appears in the song. The x-axis represents how many words are contained in a bin. Or more generally, the count of words that share the same frequency in the song. We can see that most words do not occur more than four times in "Vengeance Unleashed" compared to "Party in the U.S.A" where most words are repeated nine or less times. We can easily see that the metal songs have more words that occur once compared to the pop songs.

Now, onto Fig. 13, the total word distribution histogram comparing metal songs and pop songs is displayed. It is important to note here that both the y-axis and x-axis will not have unique word counts anymore. Given that we are aggregating all the words across the respective genres this cannot be avoided. However, we are still given interesting results. Another note referring back to Fig. 1, is that there are about 545 less metal songs than pop songs. To even the grounds a bit, the first 545 songs were removed from the pop data used upon plotting. In our sample, the highest frequency for a word in a single song for metal is 63. The highest frequency for a word in a single song for pop is 163. The distribution shape also differs where the frequency drops significantly from one occurrence to two in metal, and the pop distribution sports a much longer tail.

Considering how drastic the sound of a typical metal song is when compared to a pop song, it is reasonable to expect noticeable differences between the two genres. This may not necessarily be the case pairwise, but this preliminary metadata suggests evidence towards quantifiable distinction between "polar" genres. For our purposes, having at least a pair with significantly differing distributions indicates that the models have a chance at picking up on such details.

E. Model Building

The final step in our machine learning process is the building of our model. We will be trying to build two different models. One will be predicting based on the unmined words and the other will be predicting based on the class association rules. We wanted to build the unmined words model as a baseline to see if it is in fact the patterns that makes a good model and not just considering the lyrics of the song when classifying. We will be using the SK Learn package to build all of our model options.

Machine learning models from SK Learn package cannot take collections of words as input features. Instead, we converted our input back into transaction vector form with the help of mlxtend's TransactionEncoder function. These transaction vectors are then used as predictor variables in our models.

First, we will discuss the process of building the baseline model that predicts on the unmined words. We started by splitting the data into training and test datasets with an eighty twenty split. We then used the train dataset to train an SGD, a Gaussian Naïve Bayes, a Random Forest, a Decision Tree, a SVC, and a k-nearest neighbours model. We compared the results using balanced accuracy to begin with to decide which ones to move forward with.

Table 3. Baseline Model Initial Balanced Accuracy

Model	Balanced Accuracy
SGD	0.20787
Gaussian Naïve Bayes	0.17699
Random Forest	0.14557
Decision Tree	0.15848
SVC	0.16648
k-nearest neighbours	0.10127

As can be seen from the above table, the preliminary results were unsatisfactory, so we attempted multiple different methods to try and increase those accuracy values.

We decided to try using PCA based on our specific dataset characteristics and the project's goals. The key reason to use

PCA for this project would be to reduce input dimensions, as the number of unique words in lyrics could be very high.

Upon applying the model with PCA on our data, results were not very satisfactory. The accuracy scores for all the classifiers showed no significant improvement compared to the original model without PCA. This could be due to the original features already capturing most of the important information in the data. Decreased accuracy may have come from loss of information that occurred when reducing the dimensions.

Next, we tried resampling of the data to see if the imbalance in the genre classes was causing the low accuracy values. We implemented a random over sampler that over-samples the minority classes by picking samples at random with replacement. This process balances the data so that all classes contain the same number of samples. After applying the resampling, we tried out the models again and saw no improvement in the accuracy.

The next tactic to increase the accuracy we tried was hyper-parameter tuning. We tried tuning the hyperparameters of the top two performing models, SGD, and Gaussian Naïve Bayes. After tuning of the hyperparameters, the ones mentioned in the background section, the results were still unsatisfactory. The SGD model had a final balanced accuracy of 0.228 and the Gaussian Naïve Bayes had a final balanced accuracy of 0.267. Overall, the models predicting based on just the unmined words were not useful for classifying songs by genre. This is a good baseline to compare our frequent pattern-based model against because if the results are satisfactory then we know the frequent patterns are in fact useful for genre classification.

Next, we built models based on the class association rules we mined in the previous section. For this process we used a stratified split to split the data into training and test datasets. This is because of the imbalanced nature of the data. One step we needed to take with this model was removing the rock and electronic genres. This was due to them having very minimal instances after removing duplicate antecedents. This second model was created using the same models as the baseline model but returned much more satisfactory results.

Table 4. Frequent Pattern Model Initial Balanced Accuracy

Model	Balanced Accuracy
SGD	0.29889
Gaussian Naïve Bayes	0.54202
Random Forest	0.56441
Decision Tree	0.60647
SVC	0.38942
k-nearest neighbours	0.38437

From these results, seen in Table 4, we could already tell that the frequent patterns are much better predictors for genre than just the unmined words. Based on those results we tried tuning the hyper-parameters of the three best models: Random Forest, Gaussian Naïve Bayes, and Decision Tree. We used the GridSearchCV function from SK Learn to perform 3-fold cross validation to tune the hyperparameters.

For the Gaussian Naïve Bayes model, we provided a parameter of only the var_smoothing hyperparameter with a wide range of possible values, ranging from 0.00001 to 1.0. This resulted in the best parameter of 0.175. This grid search took less than 10 minutes to complete, which is decently fast compared to our other grid searches.

For the Random Forest we tuned the parameters n_estimators, max_features, max_depth, and criterion over a range of possible values. The best estimator returned turned out to be the default parameter values. This grid search took around an hour to complete.

The Decision Tree model was tuned on the parameters max_features, max_depth criterion, and splitter over a grid of possible values. The best estimator obtained had the max_features hyperparameter set to 'sqrt' and all other hyperparameters with the default values. This grid search had a runtime similar to the Gaussian Naïve Bayes model.

We also attempted running a grid search on the SVC model, but it took 22 hours to complete and did not result in any significant improvement on the existing models.

IV. Evaluation

Now we will discuss the final model options and compare all the results. The final model balanced accuracy values can be seen in Table 5. From these values the best model is obviously the Gaussian Naïve Bayes with a 77% balanced accuracy, but we will examine other metric values to confirm this. As well, Random Forest and Decision Tree only increased slightly compared to the initial balanced accuracy values. In Table 6 we can examine precision and recall of the models and in Table 7 we can see the F1-scores. Since we optimized the models on balanced accuracy, we will make our decision mainly on that metric. Thus, even though Gaussian Naïve Bayes has a lower precision and recall, and therefore a lower F1-score, we still accept it as our best and final model for genre classification based on frequent patterns mined from the lyrics of songs.

Another factor to take into account is the run time of the models. Gaussian Naïve Bayes and Decision Tree complete training and fitting very quickly while Random Forest takes slightly longer to fit and predict, around 5 minutes. This is another reason why we have chosen Gaussian Naïve Bayes to be our final model.

Table 5. Frequent Pattern Tuned Model Final Accuracy

Model	Balanced Accuracy
Gaussian Naïve Bayes	0.77666
Random Forest	0.56441
Decision Tree	0.61254

Table 6. Frequent Pattern Tuned Model Precision and Recall

Model	Precision	Recall

Gaussian Naïve Bayes	0.97720	0.85969
Random Forest	0.98098	0.98095
Decision Tree	0.97701	0.97061

Table 7. Frequent Pattern Tuned Model F1-Score

Model	F1-Score
Gaussian Naïve Bayes	0.91019
Random Forest	0.98074
Decision Tree	0.97314

V. Conclusion

Upon our project results, it appears that individual words by themselves are not good descriptors of genres. Yet once combined with a few or more lyrics, greater-length FPs can predict genre rather reliably. From our preliminary statistical analysis into word distributions and structure for a couple genres, a combination of words rather than unique and characteristic singletons works best for FP-aided classification.

The frequent pattern dataset we used for prediction ended up being more imbalanced than the original song-genre dataset. As a result, supports for the Frequent Patterns from the "Rock" genre were not as high as others, and the "Soul" genre generated more frequent patterns than all of the other genres in our dataset. This implies that there are more similarities in the genre's word usage compared to Rock; Rock may be more distinct in terms of lyrics and can be better classified through instrumental analysis.

Further inspection showed that most Class Association Rules obtained from frequent pattern mining contained duplicate words across multiple genres, and that there were no unique genre-defining words noticeable. This further supports our prior conclusion that individual words cannot predict song genres reliably.

Ultimately, the more simplistic, decision-based models performed the best for our project, allowing for more accurate classification. This could be caused by the data shape and categorical nature, as multi-class classification tasks are not ideal for most models.

A. Limitations

There were many limitations in our project. The main issue we faced was that the models cannot accept strings of lyrics as intended, everything must be encoded. This was unfortunate as we believe that more worthwhile results could have been gained from the pure strings.

Another issue was that once we removed the duplicate words the genre classes of electronic and rock had too few CARS instances. This resulted in us needing to remove those classes. Potentially, if we had more data that lead to more definitive frequent patterns for the rock and electronic classes and then our model could have also classified for those genres.

Several features were overlooked in the interest of computing memory. Such features like proper supports and confidence for the entire dataset's frequent patterns, mining with even lower minimum support to obtain more frequent patterns, and obtaining more samples for the original dataset in order to enhance accuracy were unable to be properly implemented due to both memory and storage limitation.

B. Future Steps

Some future steps that could be taken with this project include building a user interface for the classifier, trying a neural network, and completely more hyperparameter tuning.

If we were to build a user interface, we could create one that would accept a sentence or verse from a song and the pre-processing and mining steps would be automatically completed. Then the classifier could be implementer. This would make our project much more applicable for the general population.

Another step we could take is building a neural network. Neural networks can be very effective for natural language processing. This would also help deal with the limitation of having to encode the lyrics that we dealt with. Some neural networks that would be good options are recurrent neural networks, convolutional neural networks, or transformers.

More rigorous hyperparameter tuning on the random forest and decision tree models can be executed to see if they can surpass the current best performer, the Gaussian Naïve Bayes. By doing so, we hope to achieve even more accurate and reliable classification results.

In summary, our proposed approach provides artists and songwriters with the footwork of a valuable tool to help them compare their drafts to align with their target genre and potentially increase their chances of success in the music industry. Future work could explore incorporating audio features into the classification model to improve further accuracy.

References

[1] Bertin-Mahieux, D. P.W. Ellis, B.Whitman, P. Lamere. "The Million Song Dataset". millionsongdataset.com. http://millionsongdataset.com/ (accessed February 9, 2023).

[2] S. Raschka. "apriori: Frequent itemsets via the Apriori algorithm". *mlxtend*. rasbt.github.io. https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/apriori/ (accessed April 4, 2023)

[3] S. Raschka. "association_rules: Association rules generation from frequent itemsets". rasbt.github.io. https://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/ (accessed April 4, 2023)

[4] A. Boonyanit, A. Dahl. "Music Genre Classification using Song Lyrics". web.stanford.edu. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1214/reports/final_reports/report003.pdf (accessed March 2, 2023)

[5] J. Yang. "Lyric-Based Music Genre Classification". dspace.library.uvic.ca. https://dspace.library.uvic.ca/handle/1828/9378 (accessed March 2, 2023)

[6] R. Sadli. "Guide to Gradient Descent Algorithm: A Comprehensive implementation in Python". machinelearningspace.com.

https://machinelearningspace.com/a-comprehensive-guide-to-gradient-descent-algorithm/ (accessed April 19, 2023)

[7] Z.H. Zhou., S. Liu, *Machine Learning*. Singapore, 2021, pp. 1,159-161. [Online] doi: 10.1007/978-981-15-1967-3

[8] J. Arino. "Clustering & Classification". github.com. https://github.com/julien-arino/math-of-data-science/blob/main/slides/MATH2740-slides-09-clustering.pdf (accessed April 19, 2023)

[9] J. Filip. "pyARC". pypi.org. https://pypi.org/project/pyarc/ (accessed on March 2, 2023).

[10] S. Raschka, "mlxtend" github.io. https://rasbt.github.io/mlxtend/ (accessed on March 15, 2023).

[11] F. Pedregosa, "Scikit-Learn: Machine Learning in Python", Journal of Machine Learning Research, 2011, Accessed: Feb. 10, 2023, Available: https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

[12] T. Agrawal, *Hyperparameter Optimization in Machine Learning,* 1st ed. California: Apress, 2020, pp. 4-33, Accessed: Apr. 19, 2023. [Online]. doi: 10.1007/978-1-4842-6579-6

[13] R. Chopra, M. N. Alaudeen, A. England, *Data Science with Python: Combine Python with Machine Learning Principles to Discover Hidden Patterns in Raw Data*, Birmingham: Packt Publishing, 2019, pp. 53-54. Accessed: Apr. 19, 2023. [Online]. http://uml.idm.oclc.org/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=e000xna&AN=2204654&site=ehost-live&ebv=EB&ppid=pp_55

[14] C. C. Aggarwal, *Data Mining*, 1st ed. Manhattan: Springer Cham, 2016, Accessed: Mar. 20, 2023. [Online]. doi: 10.1007/978-3-319-14142-8

[15] J.J Aucouturier, F. Pachot. "Representing Musical Genre: A State of Art". *Journal of New Music Research.* Vol. 32, No. 1, pp. 83–9.2 003 [Online]. doi: 0929-8215/03/3201-083

[16] C. C. Aggarwal, J. Han, *Frequent Pattern Mining*, 1st ed. Switzerland: Springer International Publishing, 2014, Accessed: Mar. 20, 2023. [Online]. doi: 10.1076/jnmr.32.1.83.16801

[17] S. K. Palanisamy, "Association Rule Based Classification", Ph.D thesis, Department of Computer Science, Worchester Polytechnic Institute, May 2006. [Online] Available: https://web.wpi.edu/Pubs/ETD/Available/etd-050306-131517/

[18] F. Jiri, T. Kliegr, "Classification based on Associations (CBA) – a performance analysis," University of Economics, 2018. [Online]. Available: https://ceur-ws.org/Vol-2204/paper6.pdf

[19] B. Liu, *Classification by Association Rule Analysis*, Boston: Springer, 2009, Accessed: Mar. 25, 2023. [Online]. doi: 10.1007/978-0-387-39940-9_558

[20] A. Geron, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. California: O'Reilly Media, 2019, Accessed: Feb. 20, 2023

[21] B. Liu, *Classification by Association Rule Analysis*, Boston: Springer, 2009, Accessed: Mar. 25, 2023. [Online]. doi: 10.1007/978-0-387-39940-9_558

[22] A. Geron, *Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*, 2nd ed. California: O'Reilly Media, 2019, Accessed: Feb. 20, 2023