Song Genre Classification via Lyric Text Mining

Anthony Canicatti
Computer and Information Science Dept., Fordham University, Bronx, NY, USA

Abstract— *Text mining is often associated with the processing of the natural English language to formulate a general conclusion about a body of text. In this work, this concept is applied in an attempt to build models that classify a song's genre based on its lyrics. Such models must be able to analyze the words that compose a song's lyrics and categorize the song as one of five genres: rock, rap, country, jazz or pop. This work features classifiers such as J48 decision trees, Random Forest, k-nearest-neighbor and Naive Bayes algorithms in order to classify each song. It also discusses the significance of data collection and pre processing in order to ensure valid results particularly when dealing with string values.*

Keywords: song, genre, classification, text, lyrics

1. Introduction

A song's lyrics generally hold information about what that song is about. To perform text mining on a song's lyrics is to use data mining techniques to learn what this general meaning is given the words in the song. This work attempts to take this a step further and build a classification model that uses the results from the text mining approach to categorize a song as either a rock, rap, country, jazz or pop song. This implies a certain number of assumptions were made about songs and their genres. Firstly, it assumes that songs within the same genre typically have similar lyrics. From a high level, it is possible, and sometimes trivial, to read the lyrics of two songs and conclude they belong to the same genre. Secondly, it assumes a song's genre is mutually exclusive. Herein lies the importance of the careful selection of genres to be classified. The five genres chosen are general enough that each genre's lyrics can be classified with high precision, but specific enough that each genre's lyrics are sufficiently distant from each other. In other words, the lyrics of rap songs are sufficiently distant from those of rock songs, but the lyrics of heavy metal songs are not sufficiently distant from those of rock. For the purposes of this work, it is assumed that these five genres do not overlap.

2. Data Collection and Preprocessing

Like in all data and text mining endeavors, collecting data and performing preprocessing techniques play a critical role in preparing the data for building models. In this work, a relatively large dataset containing sets of song lyrics along with their corresponding genres is necessary to begin training and testing a classification model.

2.1 Data Collection

The primary piece of data required in this work is the lyrical content of a number of songs from each of the five genres. Lists of songs from each genre were obtained using a script that queries an online music database. The music database[1] provides tables containing artist, title, time, BPM (beats per minute) and year information. The script creates a connection to the URL pertaining to the web page that contains the information above for each genre, extracts the table values from the HTML source, and generates a list of tuples of artists and song titles. This process is repeated for each genre, and in some cases, genres are amalgamated to account for the level of granularity of the genres. For example, the database has separate entries for Rap and R&B, two genres with differing musical qualities, but relatively equivalent lyrical content. Once a list of song-artist tuples is obtained for each genre, the list is fed to a Java application which uses an open source API to retrieve each song's lyrics from an openly accessible lyric website, MetroLyrics, and stores each lyric set in a CSV file. The lyrics found on this site are all input by users, creating a few possible issues in the dataset. The lyrics may not be correct, leading to issues in preprocessing. They also may not exist at all, in which case the Java application ignores the current input and continues on with the next tuple. The result of this process is a comma-separated data file with the attributes genre and lyrics, formatted as a single string of text, for each instance.

2.2 Preprocessing

Preprocessing is arguably the most critical component given the nature of this project. If data preprocessing is done correctly, we can expect that a classification model will produce consistent, meaningful results. Before importing the raw data file into Weka to begin applying advanced preprocessing techniques, some invalid patterns for each instance were removed manually in Excel. The lyric sets found on the MetroLyrics site often contained phrases indicating repitions. For example, if a section of lyrics was repeated twice, a lyric set may contain the string "x2" preceeding it. Furthermore, lyric sets often contained labeled sections, such as "[Chorus]", or "[Verse]". These phrases were removed using Excel's Replace feature, however, it is worth noting

[1] www.cs.ubc.ca/~davet/music/

that in instances where lyric repititions are denoted with a phrase like “x2", the lyrics themselves are not repeated, even though they would be in the song. This might cause for a skew in a classification model’s results.

2.2.1 Nominal to String

Weka provides a good deal of tools to clean raw data, particularly for use in text mining applications of traditional data mining learners. The first of which required in the dataset used in this work is the Nominal to String filter. The result of data collection is a raw CSV file of lyric and genre values. Weka parses all attributes in CSV files as nominal values, so the Nominal to String filter is used to transform the lyric attribute into a string (genre is already a nominal attribute, so it can be left as is).

2.2.2 String to Word Vector

The next step is vital: the String to Word Vector filter. This filter performs three critical operations all at the same time:

1) Split each string by the whitespace character so that it is transformed into a list of words
2) From the list generated above, eliminate any ‘stop’ words which would have no impact on a learner’s classification
3) Stem each word’s ending in order to preserve only the root

In the first step, each string of words is transformed into a list and each word becomes an attribute. For example, if a lyric set is the set “hello world”, the attributes ‘hello’ and ‘world’ are created. The second step is a common text mining technique where any stop words are eliminated completely from the dataset. Stop words include basic, meaningless words that have no impact on classification, for example ‘a’, ‘and’, ‘the’, etc. Weka provides a default list of stop words, which is used in this work. The third step is also a common text mining technique. In this operation, word endings are trimmed such that their tense is not a factor in determining the meaning of the word. For example, the words “jumping”, “jumped”, “jumps”, “jumper”, etc. all mean the same thing, jump, so all these words should be trimmed so that only “jump” remains, and they all become the same word. Weka provides several different stemming algorithms that perform this stemming. In this work, the IteratedLovins Stemmer is used. Previous to applying this filter, the data consists of two attributes and a list of instances. By splitting the strings into words, the filter turns the list of attributes into a unique list of the words contained in all instances, creating a master word list for the entire dataset. The instances are turned into counts of how many times each word, or attribute, appears in the instance’s song. Take the following as a simple illustration of how this works. Suppose there are two instances of lyric sets. Table 1 displays how the dataset would look before applying the String to Word vector filter.

Lyrics	Genre
The quick brown fox jumped over the lazy dog	Rock
I have a dog and a cat, the dog is a boy and the cat is a girl	Pop

Table 1: A sample dataset prior to running String to Word Vector

After running String to Word Vector, the words in these lyrics are extracted, stemmed and stop words are eliminated. Each instance becomes a count of how many times each word appears in the lyric set. Table 2 displays how the dataset would look after applying the filter.

quick	brown	fox	jump	over	laz	dog	two	cat	boy	girl	Genre
1	1	1	1	1	1	1	0	0	0	0	Rock
0	0	0	0	0	0	2	1	2	1	1	Pop

Table 2: The same dataset after running String to Word Vector

As shown in Table 2, the String to Word Vector filter converts the data to numerical data, making it easier for a classification learner to deal with. The three operations it performs on the raw data are the key factors in determining the consistency and meaningfullness in a model’s results.

2.2.3 SMOTE

The dataset collected using the procedure described in the previous section was originally very heavily class-imbalanced, due to the very nature of the means of collecting the data. The music database contains significantly more rock, pop and rap songs than it does country or jazz. After the initial pass of data collection, the largest class, rock, contained over 1700 entries, while the smallest class, jazz, contained about 200. Weka provides a means of handling such a problem from within the preprocessing stage, the SMOTE filter. This filter balances a given class value by producing instances in an intelligent way. SMOTE uses an adaptation of a k-nearest-neighbor algorithm which analyzes instances within a class, and creates new ones based on its closest neighbors. Ultimately, the dataset was balanced into sets of 500 instances for each class, meaning 500 song entries for each genre.

2.2.4 Principle Component Analysis

The final piece of preprocessing is by far the most mathematically intensive, as well as computationally heavy. In the example used above to illustrate the String to Word Vector filter, a dataset of 2 entries is used. The original

attribute set is of size 2, where the resulting attribute set after running the filter is of size 11, the number of unique, non-stop, stemmed words. Moreover, these two lyric sets are very small. Realistically, a single lyric set is about ten times the size of the ones used in the example. Therefore, the number of attributes when running the filter against the actual dataset is overwhelming. After obtaining a dataset of 2500 instances, the String to Word Vector filter created 8922 attributes. The Curse of Dimensionality is precisely this problem: as the number of dimensions, or attributes, grows, so too does the complexity of any model, and, typically, accuracy and meaningfullness of results declines. The remedy for such a problem is Principle Component Analysis (PCA). PCA involves dimensionality reduction by transforming the matrix of data using linear algebra techniques. First, the data is centered and its covariance matrix is obtained. The trace of the covariance matrix is known as the total variance (the diagonal entries of a covariance matrix are simply the variances of each attribute). A theorem of linear algebra states that the sum of the eigenvalues of a matrix equals its trace. Therefore, each eigenvalue of the covariance matrix corresponds to some percentage of the total variance. The eigenvalue with the highest percentage of the total variance is called the first principle component; the eigenvalue with the second highest percentage is the second principle component, and so on. PCA reduces dimensionality by eliminating lesser order principle components (small percentage eigenvalues) until a desired portion of the variance is maintained. Weka defaults this percentage to 95%. Therefore, eigenvalues of the covariance matrix will be removed until 5% of the total variance has been eliminated.

PCA transformed a dataset with 8922 attributes into a weighted dataset with 1339 attributes, a significant reduction, and, though not perfect, much more reasonable than the original attribute set.

3. Experiments

For all experiments, a percentage split was used to separate training data to build models on and testing data to evaluate them. 66% (1650 instances) of the data was used to train the model, and the remaining 34% (850 instances) to test it.

3.1 J48 Decision Trees

The first experiment performed on the preprocessed dataset was building a model using J48 decision trees, with pruning. Decision trees are popular methods of building classification models because of their ease of interpretation and clarity, but may not always be the best option depending on the type of data and attribute set. In this work, the dataset contains 1339 attributes, so a decision tree generated would be so large it would be impractical to try and scan it for suspicious nodes or meaningless splits, though its results may look plausible. Nonetheless, it is interesting to use this method regardless and examine how its results differ from those of other methods that try to account for these weaknesses.

3.2 Random Forest

The next experiment performed is building a Random Forest model. This method generates a number of decision trees at random and uses them in conjunction with each other when testing the model. Using this model typically generates a higher accuracy than a single decision tree because it allows for more specific splits, though may lead to a higher chance of overfitting data because of the higher number of splits. The initial run of the Random Forest algorithm was done generating 100 random trees, then 500 random trees. In the next section a comparison of the results from these two runs is given.

3.3 k-Nearest-Neighbor

The k-nearest-neighbor algorithm is sometimes called a lazy learner, as it's means of classifying an instance boils down to evaluating the k closest instances to it and picking a class based on those k instances, as opposed to building some rule set to follow when testing. There are various parameters that allow for customizing this kind of learner. First and foremost, k itself, the number of neighbors to consider when classifying an instance. However, another interesting parameter that can be experimented with is the means of evaluating distance. Two popular methods are classic Euclidean distance and Manhattan Distance. In these experiments, k was tested with values of 1 through 5, and both distance methods were tested.

3.4 Naive Bayes

The Naive Bayes algorithm is perhaps the most popular classification model used in text mining applications. It is fundamentally grounded in Bayes' Theorem, as follows:

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

where $P(A|B)$ is the probability of an event A, given the event B. This theorem provides an excellent relationship between prior and posterior probabilities, and therefore makes for an effective technique in classification. This theorem can be extended in the following way. Suppose a vector $x = (x_1, ..., x_n)$ represents n attributes, and some event C is to be predicted. According to Bayes' Theorem:

$$P(C|x) = \frac{P(x|C)P(C)}{P(x)} \tag{2}$$

This probability is most likely a more manageable value to compute. The Naive Bayes classification model uses this

approach to classify instances. The term 'Naive' reflects that the algorithm assumes all attributes are statistically independant. In the example above, this implies that all x_i in the vector $(x_1, ..., x_n)$ are uncorrelated with each other.

4. Results

For each experiment, the primary metrics evaluated are correctly classified instances, incorrectly classified instances, kappa statistic, mean absolute error as well as confusion matrices for each method. The kappa statistic is defined as:

$$\kappa = \frac{p_0 - p_e}{1 - p_e} \tag{3}$$

Also known as Cohen's kappa, this statistic compares observed accuracy with expected accuracy, where p_0 is observed accuracy and p_e is expected. A kappa value close to 1 indicates that observed accuracy is very high (some basic algebra shows that a kappa value of 1 denotes an observed accuracy of 100%). This can be a useful metric in determining the accuracy of the specific model built compared to the actual classification. Each entry in a confusion matrix is the number of times the class in the row is classified as the class in the column. In other words, the $(i, j)^{th}$ entry in the matrix is the number of times the i^{th} class was classified as the j^{th} class. Entries along the diagonal are the number of times a class was classified as itself. In a model with high accuracy, these numbers are expected to be the largest.

4.1 J48 Decision Trees

Table 3 displays general statistics for the results of building a J48 decision tree to classify the data, and Table 4 displays this model's confusion matrix.

Correctly Classified Instances	340
Incorrectly Classified Instances	509
Kappa Statistic	0.2509
Mean Absolute Error	0.2427

Table 3: General statistics for J48 Decision Tree

Country	Jazz	Pop	Rap	Rock	←Classified As
58	6	35	21	45	Country
26	**105**	13	7	22	Jazz
57	8	**52**	27	38	Pop
27	7	29	**76**	18	Rap
47	20	31	25	**49**	Rock

Table 4: J48 Decision Tree Confusion Matrix

This model was built in 57.22 seconds, contained 319 leaves, and was of total size 637. It generated an overall accuracy of 40.05%.

4.2 Random Forest

Table 5 displays general statistics for running Random Forest with generating 100 and 500 random decision trees.Table 6 displays this model's confusion matrix for running Random Forest with 100 and 500 random decision trees.

Correct	Incorrect	Kappa	Mean Abs. Err
100 trees			
368	481	0.2916	0.2565
500 trees			
402	447	0.3412	0.256

Table 5: Random Forest General Statistics

100 trees:					
Country	Jazz	Pop	Rap	Rock	←Classified As
73	0	30	16	46	Country
29	**101**	12	7	24	Jazz
56	0	**75**	21	30	Pop
37	0	38	**70**	12	Rap
63	0	44	16	**49**	Rock
500 trees:					
74	0	37	15	39	Country
23	**101**	14	5	30	Jazz
38	0	**87**	20	37	Pop
18	0	44	**78**	17	Rap
64	0	30	16	**62**	Rock

Table 6: Random Forest Confusion Matrices

Running the Random Forest with 500 random trees yields about a 4% increase in accuracy, as 100 trees generated an accuracy of 43.35% while 500 generated an accuracy of 47.35%.

4.3 k-Nearest-Neighbor

Figure 1 displays a plot of the accuracy of the k-Nearest-Neighbor algorithm with respect to k ranging from 1 to 5. As shown in the plot, the accuracy of the model is highest at k = 1 for both distance methods. As k increases, the accuracy tends to decline and stabilize around 30%. This phenomena is somewhat expected; it is likely that instances further away from the instance to be classified are not good predictors of the instance. In other words, it is more likely that one or two of the closest neighbors to an instance are better classifiers of that instance than its four or five closest neighbors.

Table 7 below displays accuracies as well as correctly classified instances for k from 1 to 5.

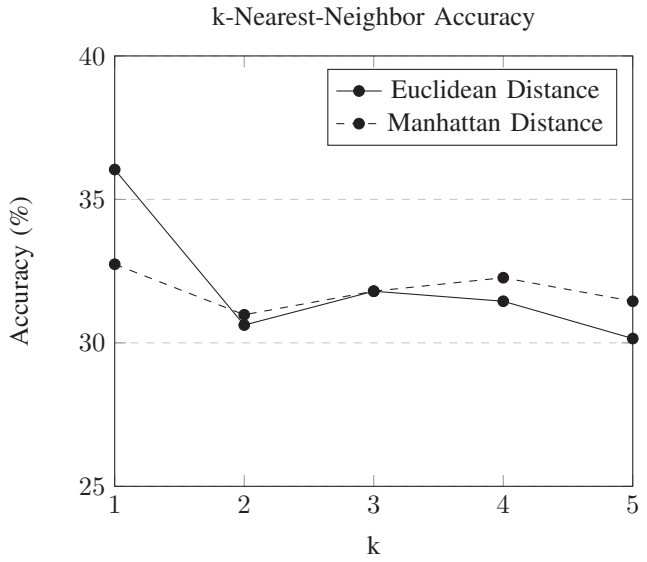

Fig. 1: Plot of kNN accuracy with respect to k

k	1	2	3	4	5
Accuracy (%)	36.04	30.62	31.8	31.45	31.15
Correctly Class.	306	260	270	267	256

Table 7: kNN accuracies and correctly classified instances

4.4 Naive Bayes

Table 8 displays general statistics for the results of a Naive Bayes classification model, and Table 9 displays this model's confusion matrix.

Correctly Classified Instances	263
Incorrectly Classified Instances	586
Kappa Statistic	0.1357
Mean Absolute Error	0.276

Table 8: General statistics for Naive Bayes

Country	Jazz	Pop	Rap	Rock	←Classified As
25	66	15	12	47	Country
21	**121**	6	3	22	Jazz
35	69	**29**	13	36	Pop
20	67	12	**32**	26	Rap
33	49	19	15	**56**	Rock

Table 9: Naive Bayes Decision Tree Confusion Matrix

The Naive Bayes classifier yielded an overall accuracy of 30.98%.

Table 10 below displays an overview of all models used. The parameters for each are those which yielded the highest accuracies. In the table, Random Forest refers to the model using 500 random decision trees and k-Nearest-Neighbor refers to the model with k equal to 1 using Euclidean distance.

J48	RandomForest	kNN	NaiveBayes
40.05%	47.35%	36.04%	30.98%

Table 10: Accuracies of all models

4.5 Results without Rap Genre

As discussed in the Data Collection and Pre Processing sections, one of the pressing issues pertaining to the dataset collected is the unreliability of the text data contained in each lyric set. Words are often misspelled, or unformatted, leading to non-distinctness in the String to Word Vector filter, as well as failure in the stemming algorithm. The rap genre is the foremost contributor to these instances of malformed or incorrect pre-processed data. To observe this genre's impact on the models' results, the genre was removed altogether and the previous experiments were performed on the data, now containing 2000 instances with 1157 attributes (after PCA). Table 11 displays accuracy results for J48 Decision Trees, Random Forest with 500 trees, k-Nearest-Neighbor with k=1 using Euclidean distance, and Naive Bayes.

J48	RandomForest	kNN	NaiveBayes
39.32%	51.69%	34.16%	34.46%

Table 11: Accuracy of Models without Rap Genre

As shown in the table above, Random Forest and Naive Bayes perform better with the rap genre removed, while J48 and kNN perform slightly worse. In the previous experiments, the per class error rates for the rap genre were 0.214 for kNN, 0.081 for Random Forest, 0.116 for J48 and 0.062 for Naive Bayes. Interestingly, Random Forest and Naive Bayes, the two models with the lowest error rate for the rap class, performed better after removing the genre, while the two with the higher rap error rates, J48 and kNN, performed slightly worse. However, it is worth noting that accuracy should expect to increase, regardless of any particular complexity within a specific class, whenever decreasing the number of class values. Considering a ZeroR approach, a simplistic model in which for each instance the most common class value is selected, the baseline accuracy is 20% for 5 class values and 25% for 4 class values, an increase in 5%. The increase in accuracy observed in the Random Forest model was just over 4%, so this increase nearly matches the expected increase when removing the class value.

5. Related Work

Music streaming services such as Pandora, Spotify, Google Play Music, etc. make use of data mining as a

means of providing subscribers personal playlists which are tailored specifically based on a user's preference in music. This is done by selecting songs a user listens to often, collecting metrics that quantify each song's musical qualities, using data mining to build a model that accurately captures the user's taste in music and providing the user with different songs that fit the model. Much work has been done using metrics that quantify things such as musical intensity, tempo, beat frequency, etc., but less have experimented with using lyrical text mining to perform a similar task. In one such work, the use of lyrics is provided *in addition* to the traditional methods to classify songs based on categories of moods (X. Hu, J. Downie, A. Ehmann). This work exhibits similar pre-processing techniques, including the removal of non-lyric text, but makes use of the lyrics in different ways. For example, it explains that function words (called stop-words in this work) actually exhibit predictive power in terms of text style analysis, and are used as an independent feature set, though yield worse results than other methods used. Furthermore, the work describes an approach called Bag-of-Words, in which lyrics are transformed into a set of unordered words, and features represent frequencies of each word, which is primarily how this work transforms lyrics into a feature set. Another approach is Part-of-Speech, in which words are grouped together based on their grammatical function in sentences. This approach provides more expressive power in that it may be helpful to analyze what type of words significantly impact classification, though may not necessarily yield better results. In another work, lyrics are used to find underlying emotional meanings in songs (D. Yang, W. Lee). This work uses 23 emotion categories such as power/strength/dominance vs. weak, active vs. passive, understatement vs. exaggeration, etc, in which songs are grouped into. Therefore, the approach used to generate a feature set is an adaptation of Part-of-Speech; rather than grouping words based on their grammatical function, they are grouped based on a pre-defined, arguably subjective, emotional function. Nonetheless, this work achieved an accuracy of 67% using an ensemble method in Weka.

6. Conclusions

The bulk of the work needed to begin performing any type of data mining experiments is contained in the pre-processing stage. As a result, the consistency and meaningfullness of the results obtained from the experiments is very much so contingent on the success of pre-processing. The results displayed in the previous section look poor at first glance, and it is without question they could be improved.

6.1 Future Work

The most glaring pre-processing error stems from the inconsistency of the lyrics contained in the rap genre (there is also a good deal of inconsistency in the rock and pop genres, but more so in the rap genre). One example of inconsistency is an instance where a word is spelled differently in two places, though each spelling neet not be 'correct'. The String to Word Vector filter only requires words to be spelled the same way, be it the correct spelling or not. In fact, as is true with text mining in general, the meaning and spelling of words is irrelevant - a model does not require a human understanding of the words in order to classify each instance. Nevertheless, it is obvious that these problems create issues for each model, and, as expected, accuracy jumps when removing the rap genre altogether. In future work, this kind of approach is optimal in attempting to achieve higher accuracies. In all experiments performed, Random Forest with 500 trees generated the highest accuracy at just over 50%. In order for all accuracies to improve, a more careful tweaking of the dataset needs to be performed. One such tweaking would be to remove all words not contained in the English dictionary (assuming the lyrics are all written in English). This approach is a bit extreme, and may not be necessary as the models themselves do not consider word meanings, but it would ease the pre-processing stage and create less inconsistency.

Another approach would be to re-evaluate the method of data collection altogether. The data is taken from an openly accessible lyric site, MetroLyrics. Given the high number of lyric sets required, it would be nearly impossible to verify the validity of each set of lyrics taken from the site and clean them to remove unwanted characters, misspellings, etc. Lastly, Principle Component Analysis clearly plays a critical role in the dimensionality reduction of text data, so it is also likely that a tweaking of this process may lead to better results. Nevertheless, it is clear that as the pre-processing of the dataset becomes more refined, the accuracies of each classification model improves.

References

[1] Xiao Hu, J. Stephen Downie, and Andreas F. Ehmann *Lyric Text Mining in Music Mood Classification*. *American music*, 2009.

[2] Dan Yang and Won-Sook Lee *Music Emotion Identification from Lyrics*. Multimedia, 2009. ISM '09. 11th IEEE International Symposium on, San Diego, CA, 2009