Genre Prediction via Lyrical Analysis

Hussain Kader and Robert Yates

December 13, 2013

1 Abstract

In an attempt to improve musical organization, we are predicting song genres by analyzing the lyrics. We use supervised learning algorithms to classify songs into one of sixteen different genres based on a bag-of-words version of their lyrics. From our 2,000 song dataset, we found that ensemble learning and logistic regression algorithms prove to be the most effective. Since the best classifier is almost eight times more effective than random guessing, analysis of a song's lyrics can be a useful tool in predicting its genre.

2 Introduction

The Tallest Man on Earth is a popular artist that can be found on essentially all major music services such as Spotify, Grooveshark, and iTunes. Even though each service catalogues exactly the same artist, they label the artist with different genres. Grooveshark labels him as a 'folk' singer, iTunes considers his music 'alternative', and at one point Spotify had featured him in a 'blues rock' playlist.

This example of differing classification highlights both our inspiration and our biggest challenge. The inspiration for this project is that through machine learning on the lyrics of a song, we can create a genre prediction service that is good enough to normalize genres across the board and greatly improve musical organization. Our biggest challenge comes from how different genres are interrelated. The same song may often fall into a number of different categories.

For our project, we focus on choosing a single 'best match' genre via analysis of a song's lyrics. We also run a secondary, follow-up experiment as an attempt to leverage our same features to predict multiple correct genres per a song.

3 Data Set

We extract our lyrics information from the Million Song musiXmatch dataset (`http://labrosa.ee.columbia.edu/millionsong/musixmatch`). For each song, we are given a dictionary that maps each word that appears in the song's lyrics to the frequency of the word in the song lyric (for example the song lyric 'hello, my beautiful friends, hello' would be given to us as `[hello:2,my:1,beautiful:1,friends:1]`). With the knowledge that the same song might be correlated to different genres based on the reference resource, all of our genres come from the same source - the Echo Nest API. Each song usually has multiple genres, so we use the first genre in the list that the API returns (since that is the most prominent genre).

Our initial genre source was the Last.fm music API, but we saw incredibly low results while using that data. Last.fm does not have genres, but instead has tags. So we wrote a script that manually hashed approximately five-hundred different tags to eleven different genres. Our use of the echonest API immediately improved our results, so we don't think that the last.fm tag groupings were as effective as that of normal genres.

Our functional dataset consists of 2,000 songs which are relatively evenly distributed over 16 genres: 'metal', 'pop', 'r&b/soul', 'country', 'punk', 'folk', 'latin', 'hip hop', 'christian', 'dance', 'electronic', 'blues', 'jazz', 'indie', 'reggae', and 'alternative'. We did not use some genres ('spiritual') that we did not have enough songs. In addition, we did not include almost 500 rocks songs. Rock as a genre is inherently very connected to other genres (ex: 'Pop Rock', 'Metal Rock', 'Folk Rock', 'Latin Rock', 'Christian Rock', 'Indie Rock', etc) and thus proved to be problematic in classifying. On average, rock shared almost 70% of words with other categories, so it makes sense that we often mistook songs that were rock as other genres and vice versa. For testing data, we randomly withheld 10% of the songs.

4 Approach

We initially implemented a baseline system using the decision tree algorithm and word frequency as our features. Our accuracy with this system was $\sim$ 29%. We then implemented a number of other features such as unique words, modified unique words with an indicator for when the word frequency is at least two, unique word count, and song sentiment and did pre-processing like removing stop words and the top 15 most common words. After that we ran our data on five other algorithms, rotating in different combinations of features. From that, we chose our top performing algorithms and did a number of extra experiments in an attempt to maximize their effectiveness.

5 Results

We tested the following algorithms using decision tree classifier as our baseline:

- Decision Tree Classifier - Uses a decision tree to classify data.
- AdaBoost Classifier - Ensemble algorithm that fits a decision tree classifier on the dataset and fits additional versions of the classifier with adjusted weights. (For each iteration, the classifier increases the weights on the data points that were correctly classified and decreases the weights on the data points that were incorrectly classified.)
- Random Forest Classifier - Ensemble algorithm that fits numerous decision tree classifiers that are each trained random subsets of the data.
- Support Vector Classification with libsvm - Separates classes into multiple binary classification problems each with a maximum margin separating hyperplane. Uses radial basis function kernel whose runtime is quadratic to the number of samples.
- Support Vector Classification with liblinear - Separates classes into multiple binary classification problems each with a maximum margin separating hyperplane. Uses linear kernel whose runtime scales linearly with the number of samples.

- Logistic Regression - Separates classes into multiple binary classifiers where the probability of a labels is computed from a logistic curve with respect to the features. (Logisitic regression is often used as an alternative to Naives Bayes since it does not assume statistical independence of features.)

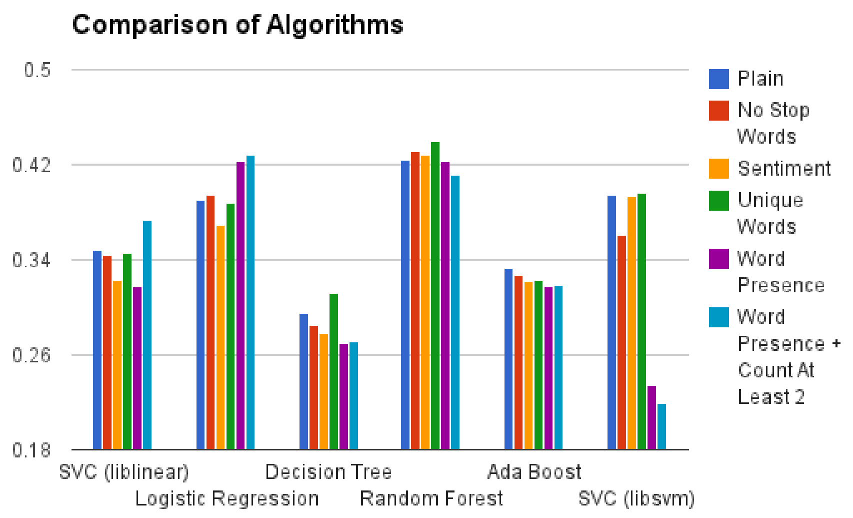


Figure 1: Comparison of Algorithms

We initially compared the accuracy of various machine learning algorithms against each other using only word count as a feature. As shown in Figure 1, the 'plain' version of random forest classifier was the most accurate at 0.425 correct predictions. The 'plain' version of decision tree classifier was the least accurate at 0.296 correct predictions.

Removing stop words ('to','a','the','its', etc.) from the feature vector improved logistic regression and random forest, however it reduced the accuracy of the other algorithms, especially SVC with libsvm. Adding sentiment (positive, neutral, negative, numeric values of how positive and negative are the lyrics) improved only random forest. Adding a feature for the number of unique words in a song lyric improved decision tree and random forest, had no effect on SVC with libsvm, and reduced SVC with liblinear, decision tree and AdaBoost. Adding a feature for word presence improved logistic regression. Adding two features for word presence and word count at least 2 improved both SVC with liblinear and logistic regression. Both word presence and word presence with count at least 2 reduced the accuracy of all other algorithms, especially SVC with libsvm.

In Figure 2, we ran different values for parameters in random forest. The parameters are: entropy (use entropy spliting criteria instead of default gini index), trees (number of estimators), max_feat (maximum number of features), min_split (minimum number of samples to split an internal node) . The 'plain' random forest classifier has 100 trees, 10 maximum features, and 2 minimum sample split.

Random forest uses the number of trees as the primary complexity parameter. We see that 500 trees has the best performance. Increasing the number of trees in general makes the classifier more accurate because more individually weighted trees means the ensemble can more closely fit the training dataset. However, algorithm accuracy starts to worsen at some point because we are overfitting to the training data and not generalizing the algorithm enough.

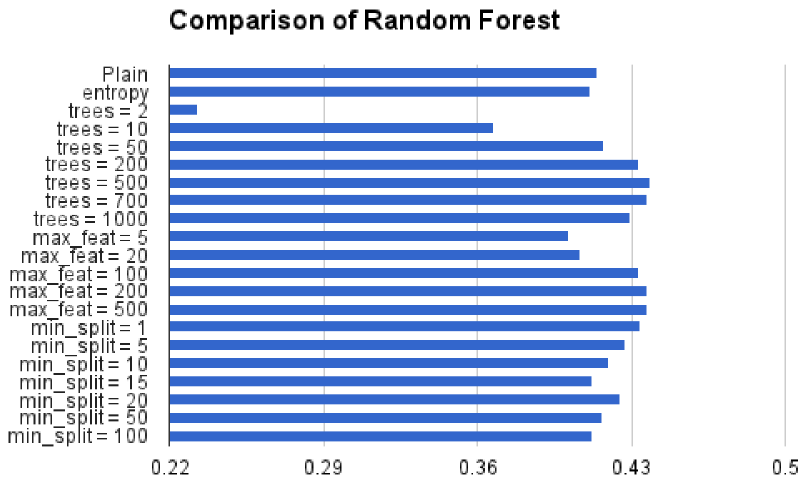


Figure 2: Comparison Of Random Forest

The maximum number of features is by default a square root of the total number of features. As expected, utilizing more features increases the accuracy score. However, increasing the number of features also greatly increases the running time of the algorithm. After about a few hundred features, additional features increase the sparsity of data, making it harder to make more accurate predictions. Random forest is most accurate with about 500 features.

Finally, the random forest algorithm uses a minimum number of samples from our dataset in order to determine the best split at a given node in each decision tree. The optimal minimum sample split is 1. In general, the less samples that are necessary, the better the algorithm performs since it does not need to be that confident to perform a split. Despite this trend, we do see an increase around 20 samples which suggests that having a large number of samples to determine a split might not necessarily be that bad if the number is set right.

We run different values for parameters in logistic regression with unique words in Figure 3. The parameters are: penality ('l2' or 'l1' norm used for penalization), dual (either true for dual formulation with 'l2' penality or false for primal formulation with either 'l1' or 'l2' penality), C (inverse of regularization strength), scaling (the value of intercept or bias term when scaling is enabled). By default, penality is 'l2', dual is false, C is 1, intercept scaling is off. Logistic regression is more accurate using the dual formulation of the L2 norm compared with the primal formulation of the L2 norm since the dual provides a lower bound on the solution to the primal. Having an 'l1' penalty decreases the accuracy of logistic regression.

Setting C to 0.1 improves the algorithm since a smaller C increases regularization strength. In logistic regression, C balances the tradeoff between fitting to the training data and fitting to future data. In general, the larger the C value, the weaker the regularization, and the more the algorithm fits to the training data. However, too large a C value and the algorithm overfits. Too small a C value and logistic regression regularizes too much and under fits the training data which is why for very small values of C (less than 0.1) the algorithm performs worse.

Enabling an intercept or bias to the logistic function improves the accuracy so long the scaling factor is set

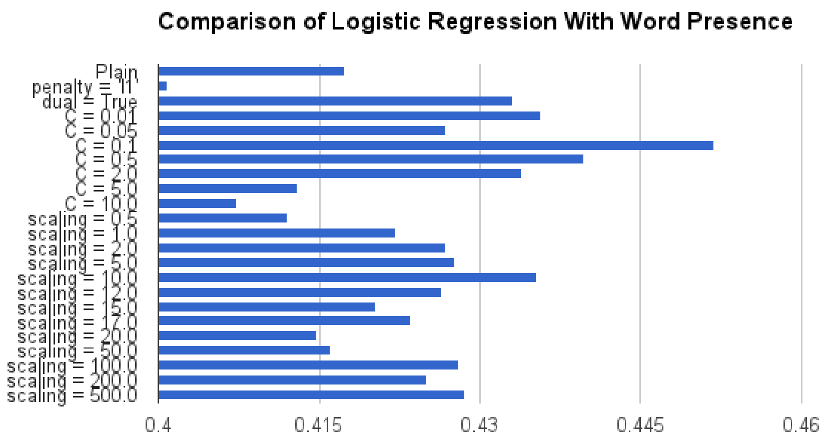


Figure 3: Comparison Of Logistic Regression With Word Presence

appropriately to the data. Here we see an intercept scaling value of 10 works best. However, large scaling values of 100 and 500 also perform well.

In Figure 4, we compare the best algorithms with the best features and parameters. RF w/ UW is random forest with unique words. $*$ mean uses trees = 500 and max_feat = 500. LR w/ WP is logistic regression with word presence. $**$ means uses C = 0.1 and scaling = 10. Dual means using the dual formulation instead of the primal formulation. WP2 means use word presence with count of at least 2. The green (unique word) and purple bars (unique word, sentiment) are not shown for random forest with unique words since these bars are computed with unique words already enabled.

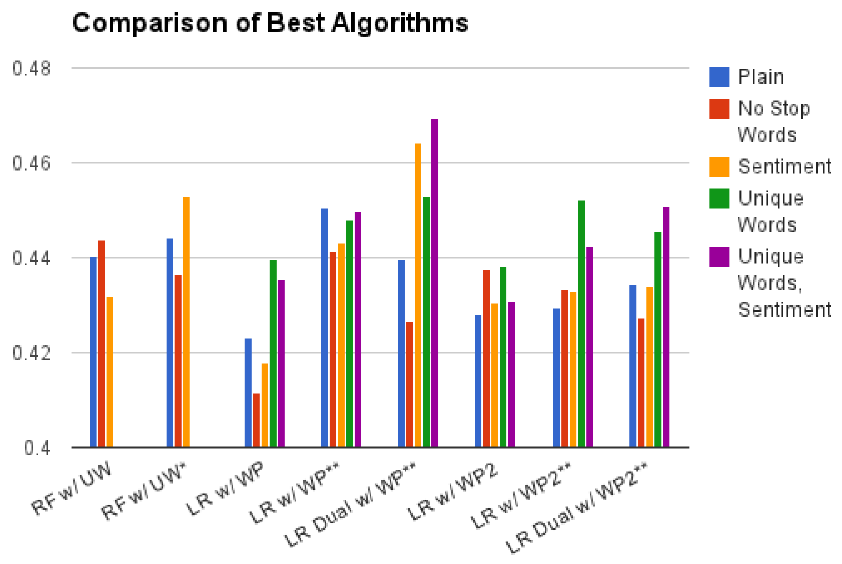


Figure 4: Comparison of Best Algorithms

Logistic regression with word presence, unique words, sentiment using the dual formulation and parameters C = 0.1 and scaling = 10 performs best at an accuracy of 0.470. The same algorithm without adding the unique words feature performs second best at 0.464 accuracy. The best random forest algorithm uses unique words and sentiment features, which has an accuracy of 0.453. We see an improvement over 4 percent using logistic regression with word presence or word presence with count at least 2 compared the standard word count feature. One explanation for this is logistic regression is able to better analyze features when they are either 0 or 1. And in general, using the optimized parameters improves the perforance of random forest with unique words, logistic regression with word presence, and logistic regression with word presence and count at least two.

6 Analysis of Genres

We decided to delve deeper into the breakdown of the lyrics in order to identify patterns as to why certain genres were classified as well (or as poorly) as they were.

Our most consistently correctly classified genre is Latin, which correctly identified almost 92% of the time, and only 4% of our incorrect labels were incorrectly labeled Latin. Most of our Latin songs contained a mix of both Spanish and English, but since hardly any non-Latin songs contained Spanish, Spanish words were a strongly weighted indicator that the song was going to be categorized as Latin. However, dance songs were classified incorrectly as Latin more often than other genres. The reason this occurred is because our dataset contained a few dance songs that also had Spanish words such as ‘amor’, ‘noche’, and ‘vida’. These dance songs most probably were a subset of Latin so were classified as Latin when they should have been classified more specifically as dance.

Another category that fared well, largely due to it’s difference in frequent words from other categories, was hip hop. Hip hop’s most frequent words are often expletives (which is not common amongst other genres), including an extremely frequent usage of the n-word which is only used (very few times) in one other category - ‘r&b/soul’. Hip hop saw a 69% success rate, it was often mistaken for dance or pop due to the consistent references to words such as ‘love’, ‘girl’, ‘want’, and ‘feel’ that the three categories share.

Very close genre pairs such as pop and dance, folk and indie, or punk and metal were incorrectly classified as each other fairly often. Pop was our category with the highest average percent of overlap (after rock which we are not using) and it shows in the fact that we only classify about 16% of pop songs correctly.

One relatively surprising result is the amount of overlap between what appear to be very different genres. For example, take reggae and r&b/soul. When listening to the two genres, the beats and song themes are quite different, but they use a large enough number of the same words to throw off the classifier. The overlap between their top 100 most frequent words (not counting stopwords) is over 70%. Some of the popular words between the two include: ‘give’, ‘feel’, ‘life’, and ‘world’. Another pair of very different genres with much larger than expected overlap is Christian and metal. Both genres consistently mention ‘god’, ‘dark’, ‘light’, and ‘eye’. It seems that the two are almost opposites in theme, but their 66% overlap is significant enough that over 18% of Christian songs were mislabeled as metal.

Adding the song’s sentiment as a feature had only a slightly positive increase to our overall results. Nonetheless, the sentiment did seem to have more significant effects on certain individual genres. For instance, our accuracy on reggae songs increased by almost 11%. Reggae songs use a lot of positive words and themes, so that became a strong indicator by which to classify them. Another genre that increased in accuracy is dance, which is also overwhelmingly positive. Dance previously had a problem in which it was often confused with pop, but in this case pop was more often neutral and dance was positive so the two were more easily distinguishable.

Sentiment also negatively affected some genres, such as metal, that we thought it would definitely help. The fact that the sentiment analyzer can only use unigram features because of the nature of our dataset is definitely a factor in this case. Metal songs also tend to use a lot of positive words in negative ways (e.g. ‘Blessed are those who run from the light’). It would be difficult to fix this issue without the use of bigram features at a minimum.

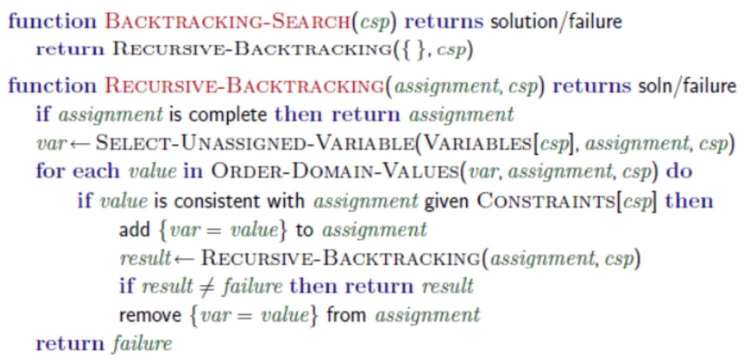
7 Multi-Label Classification

After analyzing our results from single label classification, we ascertained that a large part of our error (almost 20%) resulted from the ambivalence of songs that fall into multiple categories. We subsequently designed a process to try and combat this error.

Our idea was to model the genre-labeling as a constraint satisfaction problem (CSP). We train a single classifier for each genre that simply outputs ’[genre]’ or ’not [genre]’ for each inputted song lyric - indicating whether or not that lyric belongs in that genre. Each genre that a song is classified as by the different classifiers then becomes a variable in the CSP. For instance, the song “Cry in the Wind” by the band *Clan of Xymox* returns positive for 4 genres: ‘folk’, ‘electronic’, ‘indie’, and ‘alternative’. Therefore, the four variables in the CSP will be the aforementioned genres.

Each variable has a domain of either ‘yes’ or ‘no’. We have a map that correlates each genre with a list of genres that it can’t simultaneously be ‘yes’ with. We iterate through each genre, get the list of genres that it can’t simultaneously be with, and for each genre in that list that is a variable in our current CSP, we add a binary potential such that the two can’t both be ‘yes’. Continuing with our previous example, we would first look at folk. All the genres in our dataset that have had no correlation with indie would be mapped to it, so we would have a list [‘hip hop’, ‘reggae’, ‘r&b/soul’, ‘electronic’]. Then, we would test to see if any of our variables are in that list (in this case there would be one, ‘electronic’). Therefore, we would create a binary potential between the two making sure that there domains aren’t both ‘yes’. We would continue to do this for every variable.

There is a unary potential on each variable as well to try and push towards having more ‘yes’ genres in the results set (otherwise, without weights, every genre being ’no’ would be an optimal solution). After adding all the potentials, we run the CSP, using a recursive backtracking search basically modeled after the following algorithm:



```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)
function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Due to the fact that we do not have too many variables for any single run of the CSP, we found that we didn't need to implement any forward checking algorithms to speed things up. To measure our success we created a new data set where each group of lyrics is correlated with multiple genres (still taken from the Echo Nest API). We considered ourselves successful when every genre in our final genre set that returned with the value 'yes' was in the actual genre list. Going back to our previous example, after running the CSP we are left with the optimal solution: 'folk': 'no', 'electronic': 'yes', 'indie': 'yes', 'alternative': 'yes'. Since all three genres that returned with value 'yes' are contained in the song's actual genre list according to echonest, we have been successful.

In 150 trials where the csp is given five genres with either one or two of those genres not being correct, it provided an optimal solution that removed the incorrect genres over 77% of the time. In 500 trials of the full system, we were successful 49% of the time. The CSP performs well in theory, but because the individual classifiers often gave more incorrect labels than correct, the overall system saw more errors than we originally estimated. Nonetheless, it seems that multi-label classification via lyrical analysis is a bit more solvable of a problem.

8 Conclusion

After achieving a success rate of 47% with single-genre classification in a situation where randomly guessing would give a success rate of $\sim 6\%$, it is an inarguable fact that lyrical analysis can be used as a tool to help classify genres. We believe that having the lyrics in their original order as opposed to in a bag-of-words format would also be beneficial, as we might have been able to use more than just unigram-based features. Nonetheless, regardless of the format of the lyrics, given that the word overlap between genres is 52%, it is necessary to incorporate other facts about the song to create a truly useful genre prediction tool.

We believe that our prediction system could make a great feature in a larger system that also incorporates beat and tone analysis for predicting genres.