Classifying the Subjective: Determining Genre of Music From Lyrics

Ahmed Bou-Rabee Keegan Go Karanveer Mohan

December 14, 2012

Abstract

In this paper we address the area of multi-class classification of single-label music genres using lyrics. Previous work [1] has achieved a maximum of 40% accuracy, which we improve upon. To do so, we implement and compare a collection of supervised learning algorithms and an unsupervised algorithm on a uniformly distributed random dataset, processed from an online lyrics database to include about 5,000 songs from 7 genres. We first analyze the performance of each algorithm after preprocessing the data in various ways. We then discuss why clustering algorithms such as k-means and kNN do not perform very well in this setting. Overall we found that the ensemble classifier and multi-class Naive Bayes provide the best accuracies.

1 Introduction

Classifying music into genres based on lyrics is an interesting problem in the field of Music Information Retrieval that presents several challenges. In this paper we chose to use individual datasets with hard labels, but in reality, classifications are subjective: different people may assign different genres, or even multiple genres, to the same song. Although we chose individual datasets with hard labels, the presence of such debate among classifications speaks to a certain degree of ambiguity in the classification process. Songs in many genres may not have lyrics as well.

Despite these difficulties, however, there is still much potential for song lyric analysis. Focusing on improving the prediction value of song lyrics alone has allowed us to use music-calibrated state of the art text classification algorithms to improve upon previously established benchmarks. Liang, Gu et al., 2011 [1], had used the musiXmatch dataset associated with the Million Song Dataset [2], to classify songs into ten genres using bag-of-words lyrics features, achieving 40% accuracy. In this paper, we focus on improving lyrics-only classification in hopes that this effort can help improve music classifications by translating into even more accurate predictions when other features are considered.

2 Gathering Data

Our first attempt at this problem used song lyrics from the recently released (2011) Million Song Dataset [2]. The accuracy rates after running Naive Bayes were quite low, similar to those observed by Liang, Gu et al., 2011[1]. These authors noted that the lyrics for the songs in the database were incomplete. We believed that a complete set of lyrics, unlike what was used in other text-classifications, could cause important changes in the performance of algorithms on lyrics and hence proceeded to search for a dataset of complete lyrics.

We then used a publicly available training set [3] of complete music lyrics, with about 9,700 songs and a total vocabulary of about 44,000 words. We randomly chose $9/10^{\text{th}}$ of the data for our training set and used the rest for our test set, on which we performed Naive Bayes, multi-class SVM, and kNN. Our results were unexpected, with about 65% accuracy for L2-regularized multi-class SVM and Naive Bayes, and about 60% accuracy for k-nearest neighbors (kNN, k = 10). Upon further inspection, we noticed that our dataset was highly skewed: 55% of the songs were from the genre "classic pop and rock." In other words, if all of our test data had been assigned to this category, we would have achieved a prediction accuracy of above 50%. In an attempt to establish uniformity among the training data, we tried to manually reconstruct our data to be uniform, training 300 random songs from each genre and testing 50 different random songs from each genre. Unfortunately, our newly uniformed data was too limited to provide any reliable results. To allow a more accurate analysis of data, we decided to collect our own data via web crawling that has built-in checks to ensure a certain degree of uniformity. We wrote a Python web crawler to obtain new data from lyrics.wikia.com [4].

We took into account many factors when pooling and processing our data. While parsing the data, we ensured that only unique, English songs are considered. The genres we chose were the top 7 most popular music genres according to The Recording Industry Association of America's (RIAA) Consumer Profile [5]. Using the crawler, we pooled 800 songs from each of the following categories: blues, country, hip-hop, pop, rap, R&B, and rock. We then parsed all the lyrics to produce the vocabulary of the dataset. Each song's lyrics were converted into a multinomial event model representation that consists of a list of indices of words in the vocabulary and a list of counts for each index. In addition, several papers mentioned the use of stemming in both reducing the size of the data and helping with accuracy [6][7]. In light of this, we created both unstemmed and stemmed versions of our data sets and compared the results of our algorithms in each case. In order to stem our data, we used the Porter stemming algorithm [8], which coalesced words with common stems and removed a standard list of stop words. Our accuracy rates were similar for stemmed and unstemmed datasets, with stemming helping only to reduce the data size. Each document was represented by a matrix of 25,383 features corresponding to counts of English words.

3 Results

In this section we describe the results of the algorithms used and briefly summarize algorithms not covered in class. Each bar graph displays the accuracies of the algorithm for categories: 1.Rock 2.R&B 3.Country 4.Hip-hop 5.Blues 6.Rap 7.Pop.

3.1 Naive Bayes

We implemented the multinomial event version of Naive Bayes in which we calculated the probability of a given training example given each genre. Since we had seven classes, we had to introduce a set of

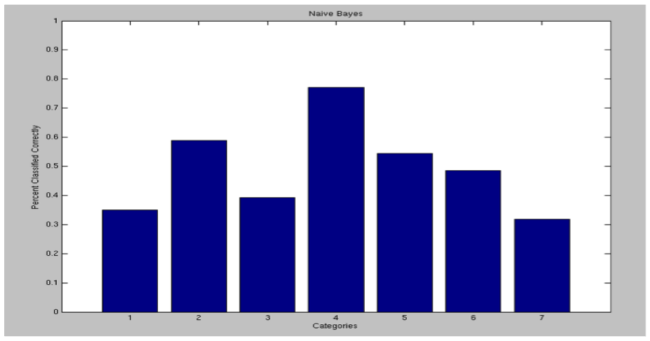

Figure 1: Naive Bayes Categorical Accuracy

parameters for each class. Thus for $l = 1, .., 7$

$$\phi_{k|y=l} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i} 1\{x_j^{(i)} = k \wedge y^{(i)} = l\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = l\} n_i + |V|}$$

and

$$\phi_{y=l} = \frac{\sum_{i=1}^{m} 1\{y^{(i)} = l\}}{m}$$

After having our probabilities and parameters trained, we calculated the probability for the test data and outputted each genre based on maximum likelihood estimates of the training data belonging to the genre.

3.2 Random Forests

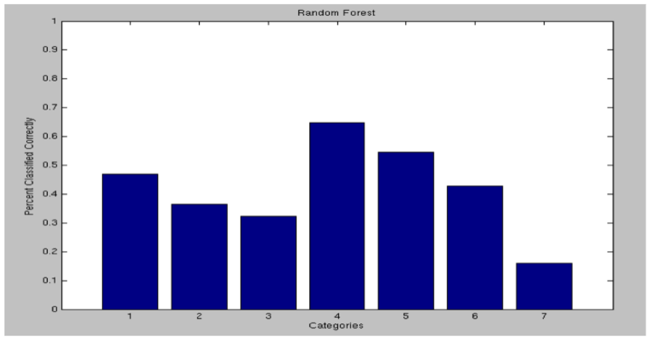

Figure 2: Random Forests Categorical Accuracy

We ran the Random Forests algorithm with 100 trees over the entire dataset and computed the error with an ooB error estimator. Random Forests classifies using multiple decision trees, each of which uses a portion of the overall data. The algorithm works by growing a given number of trees, each of which is passed a randomly sampled portion of the overall training set and some features off which to base the decision. Each node in the decision tree also randomly chooses what features to use for that particular split. To obtain a classification on test data, an example is passed to each tree, which pushes the example through it and outputs a classification. The majority classification over all trees is the resulting classification of the algorithm. Among the noted disadvantages of Random Forests is that it tends to overfit when the data contains a lot of noise [9].

3.3 Multi-class SVM

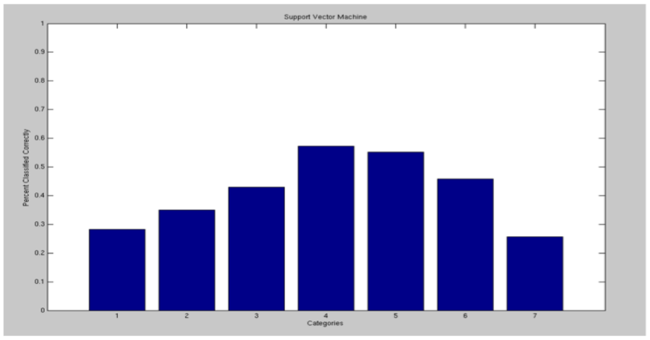

Figure 3: Multi-class SVM Categorical Accuracy

We used an L2- regularized L2-loss SVM classification [10] by implementing the "one v.s. one" method instead of "one against rest" since it is much faster and, as observed by C.W. Hsu and C.J. Lin, is comparable in performance[11]. As we were in a multi-class setting, we solved 7 binary SVMs. We determined the optimal value of the relative weighting factor, C, by trying various values, finding that the best performance was observed with C = 4.

3.4 Softmax Regression

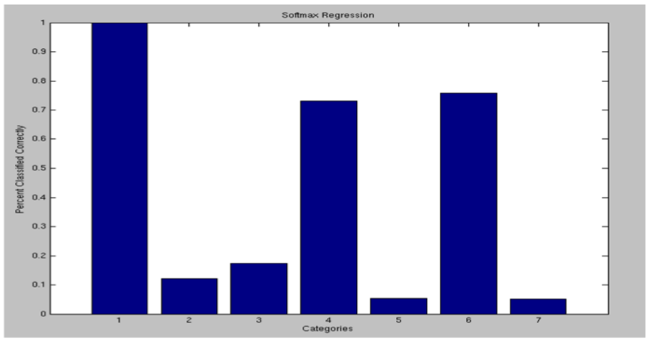

Figure 4: Softmax Confusion Categorical Accuracy

As previous existing implementations were too slow, we implemented our own version of softmax regression with regularization. In our implementation, we used the modified cost function $J(\theta) =$

$$-\frac{1}{m}[\sum_{i=1}^{m}\sum_{j=1}^{k}1\{y^{(i)}=j\}log(p(y^{(i)}=j\|x^{(i)}\ \theta))]+\frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=1}^{n}\theta_{ij}^{2}$$

where

$$p(y^{(i)}=j|x^{(i)};\theta)=\frac{e^{\theta_j' x^{(i)}}}{\sum_{l=1}^{k}e^{\theta_l' x^{(i)}}}$$

which gives the gradient $\nabla\theta_j J(\theta) =$

$$-\frac{1}{m}[\sum_{i=1}^{m}x^{(i)}(1\{y^{(i)}=j\}-p(y^{(i)}=j|x^{(i)};\theta))]+\lambda\theta_j$$

This approach has been found to work well in the area of text classification [12]. Adding the weight decay term allows for the overparameterized system to be solved without resorting to removing one of the probability vectors. Our algorithm made use of L-BFGS provided by [14] to compute the minimum of the cost function. We found that softmax initially performed very poorly with roughly 10% accuracy on both the stemmed and unstemmed data. We made improvements to the algorithm by normalizing the word frequencies per song based on the length of that song. Intuitively, this helps because longer songs would tend to have a higher word count of common words even though the higher count is not indicative of the genre. By doing this, we found that accuracy rose to about 20%. Following the idea above, we modified the original data by eliminating words that appeared in too many categories and ran the test using a vocabulary that focused on the words that were different in each set. When this was done, the accuracy rose to around 40%, comparable to the other tests.

3.5 k-Nearest Neighbors

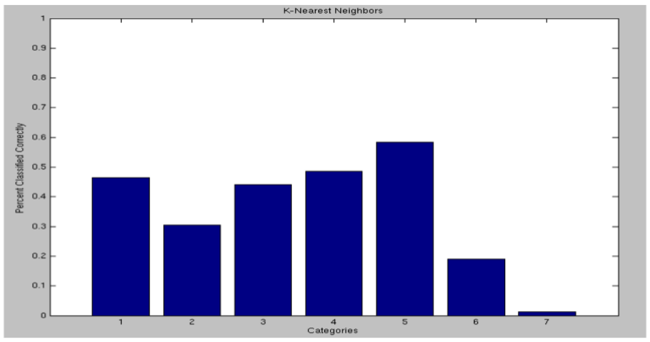

Figure 5: kNN Categorical Accuracy

The k-nearest neighbors algorithm (kNN) works by polling the points which surround a training example. More specifically, the algorithm keeps the entire training set it is given, and when a test example is given to it, it finds the k nearest points to the test example and outputs the most common class among these k neighbors. We used different metrics for finding the nearest neighbor, including euclidean distance and correlation. In general, we found that using the metric of euclidean distance gave better performance than the correlation metric. We quantitatively measured the accuracies choosing different values for k, and found that k = 250 gave us the best results over several rounds of testing.

3.6 k-Means

We decided to use an implementation of k-means [13] to test unsupervised learning on this problem. We ran k-means with 7 clusters for 100,000 iterations. Since these clusters didn't have an a priori class associated with them, we iterated over all permutations of categories and picked our accuracy for this algorithm to be from the best performing permutation. This algorithm yielded an accuracy of 26 %. Another approach we tried was running k-means with 400 clusters and replacing each data point that k-means is associated with with that cluster, using that as our new training matrix. We found that this provided comparable results with Naive Bayes and kNN. Gayathri, K., et al., 2011 [15] suggested that this implementation would increase accuracy; however, we did not see significant increases. We speculate that k-Means may cluster the songs into subcategories which are not correlated with genre.

3.7 Ensemble

Our initial ensemble of classifiers used all our individual learning methods to see which genre most algorithms classified the lyrics into. The rationale behind this was that as the number of hypotheses increases, the chances that all of them will misclassify the lyrics decreases. We observed that running this ensemble in some cases did worse than our best algorithm. This became clear since kNN and k-means were giving accuracies of around 35% and 26% whereas all the other algorithms had accuracies of at least 40% and hence, kNN and k-means were increasing the number of misclassifications. We then

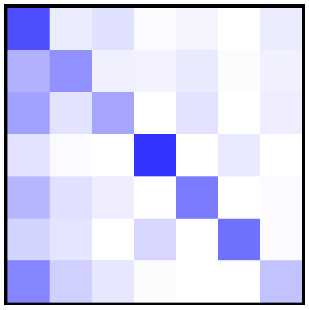

Figure 6: Ensemble Confusion Matrix: This confusion matrix represents the output of our ensemble algorithm. A bluer color indicates that more classifications were made in that square and a whiter color indicates the opposite. Category 4 (Hip-Hop) has the least confusion while category 7 (Pop) has the most confusion.

ran the ensemble with only our top 4 performing algorithms. In case of a tie, preference was given in the order of NB > SVM > Random Forests > Softmax. The updated ensemble consistently performed better than our best algorithms, reaching 53% accuracy. The reason the accuracy only improved by 3-5% in comparison to the best standalone algorithm, Naive Bayes, for any test data was because most of our algorithms were largely correctly classifying or misclassifying the same songs, again implying that certain songs cannot be classified well using lyrics.

4 Analysis

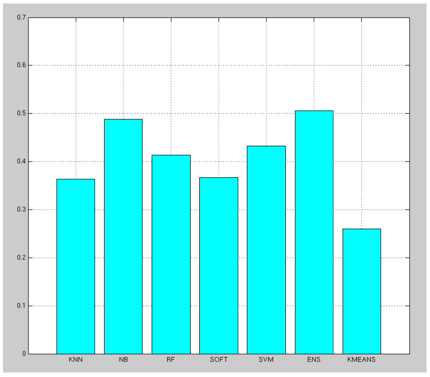

Figure 7: Each bar represents the overall accuracy of one of our algorithms over all 7 categories.

Naive Bayes consistently provided the highest accuracy rates with an average of about 48%, followed by accuracy rates of SVM, Random Forests and Softmax. We found that clustering algorithms such as k-means and kNN did not provide as high accuracies, which we believe is largely due to the overlap of the data. When we examined our dataset, we found that most music genres share at least 70% of their words, and furthermore, these words tend to be common words in the English language. Categories which contained unique words were classified more accurately than those which did not; the genre of Hip-Hop shared the least amount of words with other categories whereas Pop songs shared the most. Overall, words that are unique to a category tend to be limited to a few songs in that category and therefore do not provide much guidance in differentiating between genres. Though this affects all our categorizations in general, our clustering algorithms are in particular affected because the neighbors nearest to a particular example are often not in the same song genre.

Our best results came from the ensemble learning algorithm when it did not include kNN. Since the accuracy increase was only around 3-5%, this suggests that our models had large overlap in which songs it classified correctly and incorrectly, leading to little gain when we combined them. For instance, some songs can be remixed from one genre to others, without any change in lyrics. This means that certain songs have a certain ambiguity that cannot be learned, while others are more indicative.

5 Conclusion

We have observed how using complete song lyrics along with an ensemble of the best text-classification algorithms can help improve accuracy of music classification. While clustering algorithms have often performed well in other text-classification settings, they seem unsuited for lyrics classification. We have also observed how lyrics alone cannot provide a complete picture of a song's genre. Adding more features such as length, rhyme-scheme, and multi-word context might allow for better classification. Other possibilities of future work include experimentation with these algorithms using non-standard distance metrics. Different ways of examining the data, for example with multi-label classification, looking at the top two choices of genre and seeing if either of those match to the actual genre, may also provide novel insight.

References

[1] Liang, Dawen, Haijie Gu, and Brendan OConnor. "Music Genre Classication with the Million Song Dataset." Machine Learning Department, CMU (2011). `http://www.cs.cmu.edu/~music/dawenl/files/FINAL.pdf`

[2] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In Proceedings of the 12th International Conference on Music Information `http://labrosa.ee.columbia.edu/millionsong/`.

[3] Publicly available, processed music lyrics: `http//alliance.seas.upenn.edu/~cis520/wiki/`

[4] Free unprocessed lyrics: `lyrics.wikia.com`.

[5] Top ten most popular genres 2009 `http://76.74.24.142/44510E63-7B5E-5F42-DA74-349B51EDCE0F.pdf`

[6] Gaustad, Tanja. "Accurate Stemming of Dutch for Text Classification." Alfa-Informatica Rijksuniversiteit Groningen (2001)

[7] Jensen, Lee S., and Tony R. Martinez. "Improving Text Classification by Using Conceptual and Contextual Features." Computer Science Department; Brigham Young University (2000)

[8] Porter Stemming Algorithm included in the python library: `http://nltk.org/`

[9] Segal, Mark R. "Machine Learning Benchmarks and Random Forest Regression." Division of Biostatistics, University of California (2003).

[10] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A Library for Large Linear Classification, Journal of Machine Learning Research 9(2008), 1871-1874.

[11] Hsu, Chih-Wei, and Chih-Jen Lin. "A Comparison of Methods for Multi-class Support Vector Machines." Department of Computer Science and Information Engineering; National Taiwan University (2002)

[12] Stanford site explaining Softmax regression `http://ufldl.stanford.edu/wiki/index.php/Softmax_Regression`

[13] An implementation of k-means by Adam Coates from: `https://sites.google.com/site/k-meanslearning/`

[14] An implementation of minFunc: `http://www.di.ens.fr/mschmidt/Software/minFunc.html`

[15] Gayathri, K., and A. Marimuthu. "An Improved KNN Text Classification Algorithm by Using K-Mean Clustering." International Journal of Computing Technology and Information Security 1.2 (2011): 73-76.