Elmer Le
Quentin Moy
Jerry Zhou
Fall 2012 CS229 Final Project

Predicting Music Tags from Lyrics

1. Abstract

Currently, many music applications allow users to associate each song with a set of arbitrary words, colloquially called tags. Tags often describe the song's genre, but can also be any other attribute the user desires. One common application with genre tags is to find other songs with similar tags to generate a list of user-specific recommendations. However, we noticed a potential shortcoming of the current tag system: since all tags are user-defined, they are prone to user error and subjectivity. This raises the question, is there a better way to generate tags? This project explores ways to work around user error: is it possible to predict a song's appropriate tags given the features of the song itself? In particular, we try to utilize a song's lyrics to predict its tags.

2. Introduction

For this project, we first picked out a small list of frequently applied song tags (rock, love, pop, alternative) to examine. Then, for each selected tag, we built a classifier by retrieving training examples, each composed of a song's word counts and its associated tags. To test the classifier, we reserved another set of word counts, and used our classifier to predict whether or not an example should belong to each tag in our list. Finally, we determined accuracies by comparing the predictions to their known tags in the database. Previously, some other studies[1][2] have also tried to predict music tags using other features, like actual audio clips. They achieve roughly 60%-70% accuracy.

3. Data

For data sets, we used the publicly available (and legally produced) Million Song Dataset[3] (MSD), a large conglomeration of music data of 1,000,000 modern songs. It contains several large datasets of song lyrics, tags, artists, and several other attributes that are unrelated to our investigation. The lyrical information is already tabulated, so we were only given word counts for the 5000 most common words that appear in the songs. As a result, we were unable to examine fluency (sequences of words) in this exploration. The raw data is downloadable in SQLite databases, from which we extracted data with Python and Matlab libraries.

4. Analysis

In order to predict tags, we trained four binary classifiers that would distinguish between having and not having a given tag, given a list of word counts and their associated tags. We used Naïve

Bayes, logistic regression, random forest, and SVM algorithms to construct the classifiers. In all cases, one training examples was the list of word counts for one song, so the set of training examples would be a large set of word counts for many songs. The output was whether or not the song had the given tag. Note that, as mentioned above, the data set only has word counts for the 5000 most common words, which in most cases is enough to cover the entire song, but not always.

4a. SVM

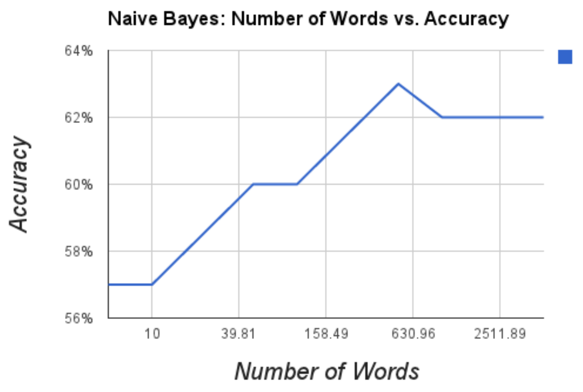
We first elected to use an SVM two-class classifier. Compared to other methods, support vector machines make few assumptions about our data and provide theoretical performance guarantees. As a tradeoff, the computation speed is highly degraded for large feature sets. Thus, we chose to only use 250 of the most common lyrics as our features, and 2000 training samples. Three classes of kernels (linear, quadratic and RBF) were tested using the SMO algorithm and all provided similar results. Below is a table of accuracies for several sample tags.

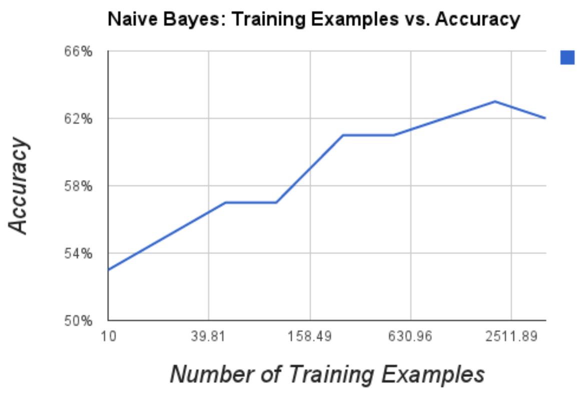
Tag	Rock	Pop	Alternative	Love
Linear SVM Accuracy	56%	57%	56%	59%
Poly SVM Accuracy	52%	51%	52%	60%
RBF SVM Accuracy	55%	56%	52%	57%

For our data, a linear kernel performed the best and also avoided the problems of overfitting and overly long computation times. This likely suggests that the relationship between our set of lyrics to tags is closer to linear.

4b. Naïve Bayes

For our Naive Bayes classifier, we experimented with using different sets of words as our features, as well as different size training sets to find what would be most effective in determining tags. The original data set contains word counts for the 5000 most common words across all songs, but many of these words actually had very few occurrences in our database, so we experimented with using only the more common words in our classifier. As can be seen from the left graph below (which used a constant of 4000 training examples), our accuracy was generally constant until we got down to a very low number of words, meaning that the less common words did not increase our accuracy significantly. We also experimented with the number of training examples that we used going from 10 songs all the way up to 4000 (with 5000 words as features). As can be seen by the graph, increasing our training size generally increased our accuracy up to around 1000 training examples. Following the parameter selection graphs is a table of average accuracies for our tags, calculated with 5000 words and 4000 training.





Tag	Rock	Pop	Alternative	Love
Accuracy	59%	60%	61%	64%

4c. Logistic Regression

For a given weight vector *w* and feature vector *φ(x)*, logistic regression uses the logistic function

$$\frac{1}{1+e^{-w\cdot\varphi(x)}}$$

to predict a binary label. In our implementation of logistic regression, due to large training set and feature vector sizes, we used stochastic gradient descent to optimize our weights iteratively.

$$w \leftarrow w - \frac{s}{t^{\alpha}}(\nabla Loss(x)),$$

where *w* is the feature vector, *t* is the iteration of the current step of stochastic descent, α is the reduction (to help it converge faster), *s* is the initial step size, and $\nabla Loss(x)$ is the gradient of the logistic loss function. The first step was to select values for these parameters, as well as the number of training examples. Given that we had enough data to have large training sets and still have test data, we optimized parameters by varying one parameter while holding others constant, running the algorithm using the different values, and selecting the one with the best test accuracy. (Granted, the general tactic for this is to use cross-validation to avoid overfitting to training data and mistakenly choosing the most overfit model. However, since we chose parameters based on test accuracy, this should not be a problem.) For the sake of space, we omit graphs of tuning parameters.

We tried to reduce the number of features in the feature vector by selecting the most significant features using forward search. However, due to technical limitations (it took impractically long to run 5000 iterations of gradient descent per feature), we opted to stick to our overall feature vector. Another technique we used to improve our feature vector was to exclude all words with 3 or fewer letters. This proved effective, increasing accuracy by several percentage points.

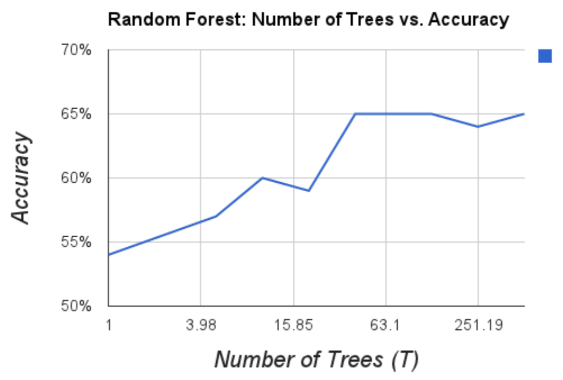
We also experimented with regularizing our weights to prevent overfitting, but given that training and test accuracies were fairly close and regularization slows the algorithm down by several times, we chose not to use it. In the end, we ran with 3200 training examples, an initial step size of 16 with a reduction of .2, and 50 iterations of descent (again, to prevent the algorithm from running impractically long).

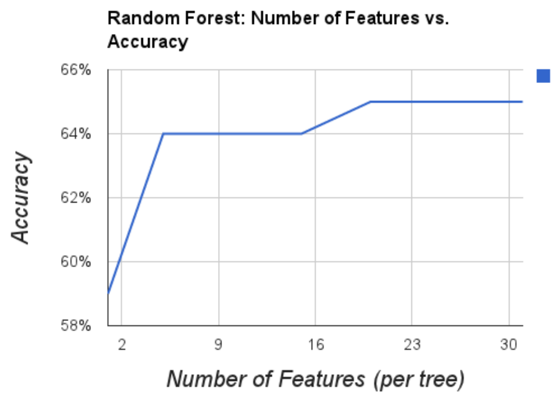
Tag	Rock	Pop	Alternative	Love
Accuracy	84%	88.5%	82.5%	87%

4d. Random Forest

The random forest algorithm is based on a majority-vote among many decision trees. For T decision trees, each one has N nodes, and we randomly choose S training examples, which we use to train that tree. For each node, we randomly choose a set of F features to use to construct the decision at that node. All trees are constructed in this way. To test, we take the test data, evaluate the result of each decision tree, and in the case of a binary classifier, take the majority vote among all trees (in the case of multi-class problem, the output is the mode of all trees).

We experimented with different numbers of trees and different numbers of features per tree in addition to the number of words when tuning our parameters for Random Forest. Testing across different sets of features gave similar results to changing the set of features when using Naive Bayes in that having too few words resulted in a less accurate algorithm, but past around 500 words the accuracies were generally pretty constant. Increasing our number of trees increased our accuracy up to around 50 trees, while increasing our number of features didn't have a very large effect on our accuracy except for in the case of 1 feature per tree. Below is a table of our average accuracies for Random Forest with 1000 words, 500 trees and 30 features per tree.





Tag	Rock	Pop	Alternative	Love
Accuracy	64%	64%	63%	65%

5. Results and Discussion

Of our four implementations, logistic regression performed the best, achieving over 80% accuracy. Previous studies[2][3] get around 60% to 70% accuracy by analyzing other features like the sound file itself, so our regression seems like a significant improvement. Our other three algorithms scored in the same ballpark as the previous studies. The inaccuracy can partially be explained by the nature of the data itself: tags are, by definition, all user-generated. There is no notion of an objectively "correct" tag. Thus, there is potential for noise to be present in all of our test and training data. Increasing training size can mitigate this problem slightly, but cannot solve it entirely. In addition, due to the creative nature of music itself, lyrics themselves are also extremely variable.

Further developing tag prediction would probably need a more holistic view of songs that takes more complex factors into account, such as the song's tempo or volume. While it would also be ideal to have a data set with objectively "correct" tags, such a dataset likely does not exist due to the nature of the tagging feature. Another approach could be using unsupervised techniques such as k-means clustering on song lyrics and generating our own tags from the top terms of each cluster.

6. References

[1] Michael Mandel and Daniel Ellis. Multiple-instance Learning for Music Information Retrieval. In Content-Based Retrieval, Categorization, and Similarity (ISMIR 2008), 2008. http://www.ee.columbia.edu/~dpwe/pubs/MandelE08-MImusic.pdf

[2] Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green. Automatic Generation of Social Tags for Music Recommendation. 2007. http://www.iro.umontreal.ca/~eckdoug/papers/2007_nips.pdf

[3] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The Million Song Dataset. In Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR 2011), 2011. http://labrosa.ee.columbia.edu/millionsong/