Abstract

Popular music streaming services like Spotify and Pandora are able to suggest songs to users based on a variety of factors. One of the major ones is the music’s genre. The aim of our project is to explore machine learning algorithms that can identify a song’s genre based on its metadata. This report outlines various research methods and discusses their performance. It provides an overview of our dataset and the performance of our final Neural Net, SVM and XGBoosting architectures. Depending on our training and test data, our Neural Net achieved between 56% - 88% accuracy, our SVM achieved between 71% - 94% accuracy, and XGBoosting achieved between 62% - 95% accuracy.

1 Introduction

In this paper, we attempt to automatically classify songs into genres based on analysis of their metadata. Music genres, such as classical, jazz, rock, and pop, are categories defined by people to group together songs with similar styles. From a practical perspective, music classification is an important task in modern society because it is the primary method for organizing similar songs. It is especially important for music applications such as iTunes, Pandora, Spotify, and YouTube which suggest songs of similar genre to their listeners. The motivation behind this project is to explore machine learning techniques that can be used to classify songs so that these applications can function more automatically. This sort of technology can be used to support various features such as the popular "radio" feature, where the application plays tracks that are similar to each other.

From a technical perspective, music genre classification is difficult because genre definitions are subjective, and songs are very complex, containing features that might require very high dimensional vectors to represent. Due to the subjectiveness of genre classification, the training data used in supervised learning techniques is usually labeled by humans and may involve certain biases. Additionally, music often contains influences from other genres, thus resulting in data that is not easily separable. These factors contribute towards creating a complex problem to address.

2 Literature Review

In our research, we found that due to the complexity of the metadata feature set for songs, many papers attempted to pre-process the data in order to improve the accuracy of their genre classifiers. One pre-processing method attempts to reduce the training set size with at worst minimal loss of classification accuracy by selectively choosing which feature vectors to train classifiers on. This method, known as pre-classification, assumes that certain feature vectors are not representative of their genre and thus should not be included in training [1]. A thorough description of the algorithm can be found in section (4.2 Pre-Classification) The classification accuracies from training on the new dataset were found to improve overall classification accuracies by 2.4% [1].

The primary motivation behind using this method is that training on large datasets could be very computationally intensive, and thus a method to reduce the training instance while maintaining the
overall classification accuracy on test data is sought. However, though the paper showed that the method was effective on large datasets, the same cannot be said about small datasets. Although the algorithm removes genre-unrepresentative feature vectors, the resulting dataset set can be substantially smaller than the original training set, which can drastically increase the classification margin of error on small datasets. Additionally, the resultant dataset after pre-classification does not guarantee a uniform distribution of feature vectors from each genre. Thus, it is possible that the pre-classified dataset has a skewed distribution, which introduces genre over and under representation, negatively impacting classification accuracies.

A very popular classification method which has shown to produce state-of-the-art results is XGBoosting [2]. XGBoosting utilizes tree boosting, a highly effective and widely used machine learning method to significantly improve algorithm accuracies [2]. Like other tree ensemble methods, given a dataset, XGBoosting uses additive regression tree functions to predict an output. Each tree stores continuous weight values throughout its leaves which represent their score and are used to calculate the prediction score of the tree. The prediction output of an instance then is the sum of all prediction scores from all the trees. To reduce potential overfitting problems, XGBoosting implements shrinkage, which scales newly added weights by a factor of $n$, and column subsampling, which is used in RandomForest and in TreeNet for gradient boosting as effective reduction methods [2]. Additionally, the method also supports popular split finding methods such as Basic Exact Greedy Algorithms, Approximate Algorithm, and Sparsity-Aware Split Finding, and implements cache-aware access to optimize computation complexity in split finding with large datasets [2].

XGBoosting is a very novel and effective machine learning method that is used in numerous data science competitions with incredible success [2]. A large problem the method overcomes is that its loss function includes functions as parameters and therefore cannot be optimized using traditional optimization methods in a Euclidean space [2]. However, by training the method in an additive manner and taking the second order Taylor approximation of the loss function, we find that only the gradient and second order gradient statistics of each leaf are needed to optimize the function [2]. This is an impressively simple solution to the problem. Additionally, XGBoosting builds upon the deficiencies of other tree based ensemble methods such as RandomForest, which is tends to overfit, by including multiple regularization techniques. Furthermore, XGBoosting notably takes large datasets into account by implementing cache-awareness split finding to optimize split finding performance when the CPU cache is not large enough to fit the method’s gradient statistics.

3 Dataset

The dataset we used is called the Million Song Dataset (MSD) [3]. It is a free database that contains audio data for a million contemporary songs. The data itself is not the raw audio of each song, but instead is precomputed metadata for each song. The (3.2 Features) section provides more in-depth details about this metadata.

3.1 Quality

The metadata for each individual song provides us with a wide variety of information, such as the song length, speed, harmonies, and key frequencies [4].

However, there are a couple of significant issues with the dataset as it stands. Firstly, some of the genres in the dataset seem very closely related to others, which could cause a reasonable amount of overlap e.g. "metal" / "punk", "classic pop and rock" / "pop" etc. Therefore, we needed to use a subset of the data such that each genre would be distinct from the others.

Secondly, the distribution of songs over all genres is incredibly uneven, as illustrated by Figure 1. As shown, classic pop and rock has almost 6 times as many data points as dance and electronica. If we moved forward with the data as it is, our architectures would be heavily biased towards the categories that dominated the dataset. Therefore, we needed to select a subset of the data such that the distribution of samples across genres was more even, while still having enough data points to properly train our architectures.

Because this dataset has been crowd-sourced, one key point to note is that the genre labels were provided by humans. Music genres are incredibly subjective, and thus this dataset may exhibit certain biases and preferences.
Keeping these issues in mind, we selected the three following genres to analyze further (the number of songs for each genre are also shown):

- dance and electronica (4935)
- jazz and blues (4334)
- soul and reggae (4016)

3.2 Features

The MSD provided us with metadata and a genre label for each song, from which we selected the following salient features [4][5]. Each song was represented as a 30-dimensional feature-vector:

- loudness
- tempo
- time signature
- key
- mode
- duration
- 12 timbre means (MFCC-like)
- 12 timbre covariances (MFCC-like)

3.2.1 Mel Frequency Cepstral Coefficients (MFCCs)

Different genres tend to involve different styles of music and sound, and it is useful to be able to distinguish between these sounds. Timbre is the term used to describe a sound independently of pitch or volume [6]. Mel Frequency Cepstral Coefficients, or MFCCs, are representations of the short time power spectrum of a sound [7]. The envelope of the short time power spectrum of the audio signal is a function of its source, and MFCCs allow us to approximate this envelope. Because of this, MFCCs are widely used in speech recognition because they are able to detect specific phonemes by approximating the shape of the speaker’s vocal tract. We used MFCC-like features for music genre classification because they may provide information about instruments or vocal styles present in each track. The Mel scale is useful in music/speech classification problems because it maps a measured frequency to the perceived frequency, to give a better representation of how a human perceives the sound. Our dataset includes MFCC-like timbre data, which encapsulates the mean and covariance of 12 timbres for each song. Figure 2 shows how these MFCCs are computed [7].
4 Data Pipeline

4.1 Visualization I

This section focuses on our initial visualizations of our dataset. Since each data point was represented as a 30-dimensional feature vector, it was difficult to get a high level understanding of what the data looked like for each genre. To visualize this, we created a histogram for each of the 12 timbre means (see Figure 3). From the graphs, we observed that it was difficult to draw boundaries between genres since they all fell into similar distributions. Therefore, our algorithms were likely to struggle to classify the data. To address this problem, we looked into pre-classification techniques to improve the quality of our data.

![Figure 3: Distribution of each of the 12 timbre means for 6 distinct genres](image)

4.2 Pre-classification

In our pre-classification stage, we implemented an instance selection method reviewed in section (2 Literature Review). The method aimed to reduce the size of large training sets while improving overall classification accuracy on test datasets. The method described in the paper reduced the training set by selectively removing noisy or unrepresentative data from the training set. While our dataset was small (~13,000 instances), our initial data visualizations revealed that features across genres were not very distinctive from one another. This suggested that there were many genre-unrepresentative or noisy feature vectors that would negatively impact our training set. Thus, we thought it would be useful to implement this method and measure its effectiveness on our dataset.

The algorithm described by the method involved iteratively training and testing two classifiers. The data was first divided three uniformly distributed folds, each with the same proportion of instances from each genre. Then an 3-class classifier was trained on the first fold and tested on the second. After testing, all the misclassified instances of the second fold were removed from the data and a new 3-class classifier was trained on the remaining data from the second fold. The new classifier was then tested on the third fold, and once again all the misclassified data instances were removed from the third dataset. These steps were repeated three times in total with the following substitution of folds: 1-2-3, 2-3-1, 3-1-2 [1] before obtaining our pre-classified dataset.

4.3 Visualization II

After pre-classifying our data, we expected to see a greater distinction between genres across features. Using two dimensionality-reduction techniques, namely Principal Component Analysis and Locally Linear Embedding, we plotted the resulting differences between the original and pre-classified datasets (Figure 4).

4.3.1 Principal Component Analysis (PCA)

Principal Component Analysis is a technique that takes a set of high-dimensional data points and maps it to a lower-dimensional representation of the data by maximizing the variance of the projected data.
and minimizing the mean squared distance between each data point and its projection \[8\]. Importantly, PCA models linear variability in data. In our case, we reduced our 30-dimensional feature vectors into 2-dimensional vectors so that we could easily visualize them in Cartesian coordinates (Figure 4).

Similar to our discoveries from our initial visualizations (4.1 Visualization 1), in the original data, all three genres merged together to form one big cluster which made it very difficult to classify songs correctly. The progression showed how each additive genre overlapped substantially with the previous graphed genres. The second genre (green) significantly overlapped with the first genre, and then the third genre (red) considerably overlapped with both the red and green genres. In the pre-classified data, while the three genres were still clustered together, it was much easier to distinguish between them in each plot compared to the original data. After pre-classifying, it appeared that all the red points that overlapped with blue were discarded, thus making it relatively easier to distinguish between the blue and green genres. When the red genre was added, while there was significant overlap with the green genre, there seemed to be a much better defined separation boundary between the red and blue genres compared to the same plot with the initial data.

Even though it looked like the data was now more separable than before, it is important to note that after pre-classification, we were left with significantly fewer data points than the original data, especially for a dataset as congested as this. Therefore, the advantages gained by pre-classification may have been countered by the disadvantage of having a much smaller training set.

4.3.2 Locally Linear Embedding (LLE)

As mentioned above, PCA is an effective dimensionality reduction technique to model linear variability in data. However, because our data is so messy, we also wanted to explore a technique that would allow us to model nonlinear variability in our data. Locally Linear Embedding, or LLE, is an algorithm that assigns some number of neighbors to each data point. It computes weights that best reconstruct the original data from its neighbors and uses those weights to compute a new set of vectors that are known as embedding coordinates \[9\]. These embedding coordinates have lower dimensionality than the original data.

After performing LLE, we noticed that the general shape of the cluster was almost identical between the two datasets. It was interesting to note that the pre-classified data was dominated by points from the red genre, which may have skewed the output.

Based on our results, it seems clear that PCA with pre-classified data gives us the best visualization where we can distinguish between genres. The dataset we began with did not provide us with very cluster-able data. Pre-classification improves this, but only to a certain extent. While we would
expect pre-classified data to give us better results, we need to be wary of the fact that the number of datapoints in our pre-classified dataset was significantly fewer than that in our original dataset.

4.3.3 K-means Clustering

Another method we used to classify the metadata into genres was K-means clustering with squared Euclidean distance. This helps measure the relative effectiveness of the pre-classifying method, and also how unique the different genres are.

As shown in Table 1, each of the three clusters consisted of approximately one third of each genre, which is the expected cluster if we randomly distributed the data. This reinforces the notion that the metadata for different genres overlap significantly. The pre-classified data improved the clustering for Soul and Reggae, but Jazz and Blues and Dance and Electronica performance decreased as the distributions became closer to one third. We noted that the pre-classification method caused data points for Soul and Reggae to decrease significantly, from ~30% of the original data, to ~8% of the pre-classified data. This may hinder classification and clustering since there is not enough data for that genre.

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Raw Data</th>
<th>Pre-classified Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Jazz and Blues</td>
<td>Soul and Reggae</td>
</tr>
<tr>
<td>Cluster 1</td>
<td>33%</td>
<td>36%</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>38%</td>
<td>36%</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>29%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Table 1: K-means Clustering of Raw and Pre-classified Data

5 Classification Algorithms

5.1 Neural Network

One classification technique we used was a neural network for multi-class classification. Neural networks can be effective in learning hard-to-discriminate patterns in datasets. We believed that Neural networks would be able to determine which features were correlated with different class labels given our complex feature set. Neural networks are comprised of neurons which take in inputs and multiply them by unique weights. The results are then passed through an activation function which sums the weighted inputs to obtain a prediction. A single layer neural network features some number of neurons. Multilayer neural networks feature a chain of such layers in which certain outputs of one layer act as inputs to the next layer. The goal is to learn the weights that provide the highest classification accuracy. Neural networks are trained in an iterative manner in which each iteration may change the weights as the network cycles through the training data via a method known as back propagation [10]. As a result, neural networks can be expensive to train, both in terms of computation and time. In order to classify a datapoint, the trained neural network takes in the datapoint and outputs the predicted class label.

After some experimenting, we settled on the following architecture for our neural network, implemented using the Keras Python package, built on Tensorflow:

- 5 dense hidden layers
- 50 neurons per layer
- ReLU activation function
- Softmax activation for output layer

We noticed that fewer hidden layers resulted in better accuracy. Between 1-10 hidden layers, we found that having 5 hidden layers performed marginally better than all others.

5.2 Support Vector Machine (SVM)

Another classification technique we tested was the Support Vector Machine. This algorithm is designed to find an optimal hyperplane separator between two labeled datasets that maximizes the
margin between the two datasets. SVMs are effective for high-dimensional data while also being more memory efficient and faster to train than multilayer neural networks.

SVMs are usually implemented for binary classification problems but can be expanded to perform multi-class classification as well, using two different techniques. The one-vs-one technique trains a unique SVM per unique pair of classes. A datapoint is then classified by each of these individual SVMs, and the class chosen most often is chosen as the final label. Given $N$ classes, this results in $\frac{N(N-1)}{2}$ unique SVMs being trained. Another technique is the one-vs-all technique, which trains a unique SVM per class label. This method uses continuous values based on the SVM decision function. In order to classify a datapoint, it determines which SVM results in the highest confidence and chooses that class. This requires training only $N$ SVMs and thus is less computationally expensive than the one-vs-one method [11]. In addition to exploring the different multi-class classification techniques, we also explored four different kernel functions: linear, polynomial, RBF, and sigmoid.

We implemented a hard SVM using the Sklearn Python package. After experimentation, we also settled on using the one-vs-all training method as it consistently performed on par with or better than the one-vs-one method.

5.3 XGBoosting

The third classification technique we used was XGBoosting. As described in the relevant literature section, XGBoosting employs gradient tree boosting to greatly improve classification accuracies in many large-scale datasets and challenges. Despite having a small dataset, we wanted to test the effectiveness of tree based ensemble methods on our data, and XGBoosting was a natural choice because of its improvements on pre-existing ensemble methods and its incredible success with other datasets across different competitions [2].

Mathematically, XGBoosting can be described as an iterative additive regressive tree algorithm of two main steps (at every iteration, XGBoosting adds a new tree into the objective function). The first is optimizing a regularized L2 loss function at time step i given data, and the second is calculating the optimized function’s quality score given the optimum weights and training data. To derive the optimum objective function weights, the algorithm needs only the gradient and second order gradient statistics on each leaf of the tree. These gradients are input into a simple scoring formula (also used to calculate the quality of a tree) to calculate the optimum weights, where the closer the output is to zero the better the tree is, and update the objective function accordingly. Then, given data, the algorithm sums all the leaf weights across all the tree to obtain a prediction. These outputs are then used to calculate the overall objective loss using the algorithm’s regularized L2 objective loss function, and the process is repeated with an additional tree through k-iterations.

We implemented XGBoosting using the XGBoost Python package and tested on three package supported classification methods: GBLinear, GBTree, and DART. GBLinear. GBLinear fits data using a gradient boosted linear model, GBTree fits it using a gradient boosted tree based model, and DART combines Dropout with Additive Regression Trees to reduce overfitting when fitting the data.

6 Performance

After testing, the various algorithms had mixed results, with classification accuracies ranging from 53% to 95% based on the data and algorithm used. Our testing framework involved the following training/testing scenarios:

- training and testing on all data (3:1 test/train split)
- training and testing on pre-classified data (9:1 test/train split due to small size)
- training on pre-classified data and testing on all data

Our results are summarized in Table 2. The highest baseline accuracy we achieved for training and testing on all data was 77% using the linear SVM. Other algorithms performed slightly worse.

We also tested the effectiveness of performing pre-classification. After training and testing on the pre-classified data, our accuracy jumped as high as 95% with the DART and GBTree models and 94% with the linear SVM. This shows that our pre-classification was successful in filtering out noisy datapoints, thus resulting in a dataset that has more distinct and separable genre clusters. We
### Table 2: Final Results

<table>
<thead>
<tr>
<th>Train / Test</th>
<th>Neural Net</th>
<th>Hard SVM (one-vs-all)</th>
<th>XGBoosting (100 iterations)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Linear</td>
<td>Poly</td>
</tr>
<tr>
<td>All / All</td>
<td>58%</td>
<td>77%</td>
<td>54%</td>
</tr>
<tr>
<td>PC / PC</td>
<td>88%</td>
<td>94%</td>
<td>64%</td>
</tr>
<tr>
<td>PC / All</td>
<td>56%</td>
<td>71%</td>
<td>53%</td>
</tr>
</tbody>
</table>

All / All = Train and Test with original data (3:1 split), PC = pre-classified data

saw significant improvement (between 10% to 30%) in the classification accuracy for almost all algorithms for this scenario.

We also explored the idea of training on pre-classified data and testing on all data to analyze whether training on the pre-classified data can improve our classification accuracy for the entire dataset. However, we found that our accuracies dropped slightly for almost all algorithms (the only exception was XGBoosting’s DART method, which saw a 12% improvement). This is not surprising as pre-classification filters out datapoints that are closer to the boundaries between genres, thus resulting in a dataset in which the datapoints are very representative of their genre. Models trained on the pre-classified data do not see the "noisy" data and therefore develop decision boundaries that are "too good" for real-world data.

### 7 Conclusion

In this project, we attempted music genre classification using metadata from the Million Song Genre Dataset (~59,600 data instances). Visualizing our dataset using histograms, PCA, LLE, and K-means clustering on the raw data revealed that data from many genres overlapped, suggesting that our data was not distinctive by genre, or necessarily representative of its respective genres. Additionally, we found the dataset exhibited heavy skew in the distribution of data from each genre, with genres like ‘classic pop and rock’ forming 1/3 of the dataset. Thus, in order to avoid genre over-representation we narrowed our genre selection to three uniformly distributed genres: ‘Jazz and Blues’, ‘Soul and Reggae’, and ‘Dance and Electronica.’ We then attempted to clean our data by implementing cross fold pre-classification to filter out noisy and genre-unrepresentative data. While this produced a dataset with more genre distinctive data instances, we found that in most cases it reduced our classification accuracies on the test data. We ran three classification methods (Neural Network, SVM, and XGBoosting) to train and test our datasets (original and pre-classified) on our data with varying accuracy. Almost all of our classifiers produced better results when trained and tested on all the data from the three genres than when trained using the pre-classified data, achieving in peak accuracies of reached 58%, 77%, and 76% for Neural Networks, SVMs, and XGBoosting respectively. Interestingly however, we found that DART resulted in substantially better classification on Pre/All data vs All/All data with a margin range of 12%.

### 8 Future Work

In the short-term, we would like to explore the XGBoosting DART classification method further to determine why we see an improvement in classification accuracy when training on pre-classified data and testing on all data compared to training and testing on all data.

After much analysis of the Million Song Dataset, we found that we were very limited by the metadata it provided. If we had access to a larger audio library with raw audio data, more information about the music tracks could be extracted from the audio source. One popular data extraction technique is known as frame analysis, in which very short audio clips (frames) are sampled from the audio and used as the data for training and testing. We can also experiment with different pre-classifying methods such as dynamic frame analysis, which dynamically tunes supplementary parameters associated with frames to improve classification accuracy [12].

Another area we could investigate in the future is classifying lyrical music with lyrics. The MSD provides data about song lyrics with a Bag of Words model, and this data could be used to further classify lyrical genres.
References


