Genre Classification of Spotify Songs using Lyrics, Audio Previews, and Album Artwork

Tyler Dammann and Kevin Haugh, *Stanford University*

***Abstract*—This paper is an attempt to attack the problem of genre classification of music from a variety of angles. Three different types of data (song previews, album artwork, and lyrics) are used to train three models (a Recurrent Neural Network, k-Nearest Neighbors, and Naive Bayes, respectively) and the outputs of the three are again combined to classify a given song. The combined model was able to achieve 91.75 percent accuracy on the test set.**

I. Introduction

Genre classification in music is a difficult and highly subjective issue, and in many cases, even humans disagree on which genre a song falls into. Currently, Spotify is doing some interesting work classifying music into over 1,300 very specific micro-genres based on song metadata [1]. However, little work has been done on using other types of available features for genre prediction. In this paper, we aim to predict genre based on a song's lyrics, audio waveform, and album artwork.

We use a Naive Bayes model on lyrics, a Recurrent Neural Network on audio waveform data, and a k-Nearest Neighbors approach on our album artwork. We also explore different model stacking techniques, including weighted sums, linear regression, and neural networks to combine the results of our three models and improve our overall performance.

II. Related Work

Other CS229 projects in the past have explored genre classification. A paper authored by CS229 students Michael Haggblade, Yang Hong, and Kenny Kao explores the most effective learning algorithms for genre classification using audio waveforms [2]. In their paper, they found that using a neural network trained on songs' MFCC features resulted in a higher accuracy than other common models. We will be using a similar technique on our audio data, as they found it to be highly effective. Several other papers attest to the usefulness of MFCCs in audio analysis, including *Mel Frequency Cepstral Coefficients for Music Modeling*, which determines that MFCCs are very effective at summarizing audio waveforms of songs [3].

In another paper published in the Transactions of the International Society of Music Information Retrieval, the authors discuss new techniques in generating features for genre classification from song lyrics [4]. They use sophisticated methods to extract information about rhyme scheme, parts of speech, and take punctuation into account. They found these features to be much more useful than a bag of words feature set.

We were unable to find significant work on the use of album artwork for image classification.

III. Dataset and Features

A. Data Sources

The initial lyric data is taken from a dataset from Kaggle [5], and the album artwork, audio waveforms, and genre labels for each song were downloaded using the Spotify API. The final dataset consists of 4,000 songs in the genres Christian, Metal, Country, and Rap, and are split 80/10/10 into train, test, and development sets. These four genres were chosen in particular because they are more easily differentiable (using different instruments, different themes, etc.) than other possible genres.

One small challenge is the decision on how to determine which large genre a song belongs to. The Spotify API returns a list of 5-6 "micro-genres," rather than a single definitive genre. An example might be "Christian rock" or "pop rock." Thus, in order to determine which genre a specific song belongs to, we defined the following metric:

$$G(x) = \text{argmax}_y \Big(\sum_{i=1}^{n} g_y(x_i) \Big)$$

Here, x is the vector of micro-genres for a given song. For each genre y in our predetermined list (Christian, Metal, Jazz, and Rap) we look at each micro-genre x_i and compute $g_y(x_i)$, whether the large genre is a substring of the micro-genre. We then find which one of the large genres has the most instances and classify the song as that genre.

B. Model One: Lyric Data

The lyric data from the Kaggle dataset was only minimally cleaned when we found it. We thus parse and clean the lyrics by removing delimiters and non-alphabetic characters. Then, we remove stopwords and words not in the English dictionary. Finally, we create bag of words features for each song, in which each is represented by a vector of length n, where n is the number of unique words present in the lyrics in all of the cleaned songs. Each feature $x^{(i)}$ for $i \in \{1...n\}$ represents the number of times the word i appears in the specific song.

C. Model Two: Sound Data

After using the Spotify API to acquire the 30-second previews for each song, we trim the previews to the middle 10 seconds. Then, frames are extracted every 23 milliseconds, and MFCC features are taken at each frame. Mel-Frequency Cepstrums (MFC) represent the power spectrum of a sound, or the intensity of the sound at varying frequencies. Mel-Frequency Cepstrum Coefficients (MFCC) are the amplitudes of the MFCs. Fig. 1 [6] represents a spectral density plot of a

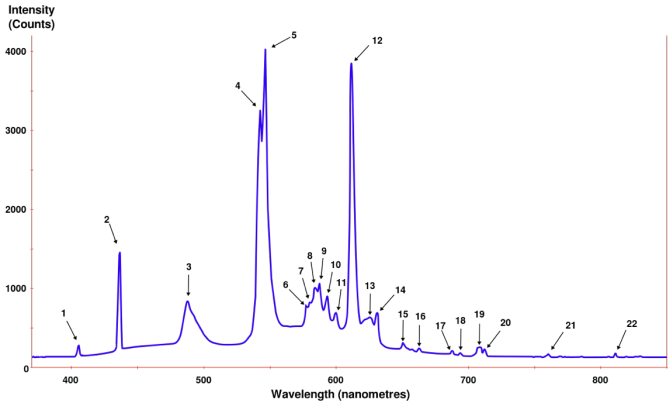

Fig. 1. Sound Spectral Density Plot

Fig. 2. Example Metal Song Artwork

wave at a specific point in time, in which the wave has been decomposed into a sum of different waves of varying frequencies. The numbers show what the MFCCs are representing - the intensity of the component waves at different frequencies.

At the end of this process, each song preview is represented by a time-series vector, where at each time step the waveform is represented by 13 coefficients.

D. Model Three: Image Data

The 64x64 pixel album artwork acquired from the Spotify API is converted to image matrix representations, from which we extract a feature vector of 4,096 pixels values for each song. Then, Principal Component Analysis (PCA) is used to reduce the number of covariates to around 3,200 in order to later perform Linear Discriminant Analysis.

IV. Methods

A. Model One: Naive Bayes on Lyric Data

For our lexical model, the bag of words features taken from the lyrics of each song are used to train a Naive Bayes model. The Naive Bayes model assumes the features x_i are independent, meaning the likelihood function takes on the form:

$$p(x_1, ..., x_n|y) = \prod_{i=1}^{n} p(x_i|y)$$

The model learns the paramaters $\phi_{i|y=j}$, representing the probability of the number of times word i is included in the lyrics given the song comes from genre j, and ϕ_y, representing the probability of a some coming from a certain genre, and uses Bayes rule to predict $p(y|x_1, ..., x_n)$.

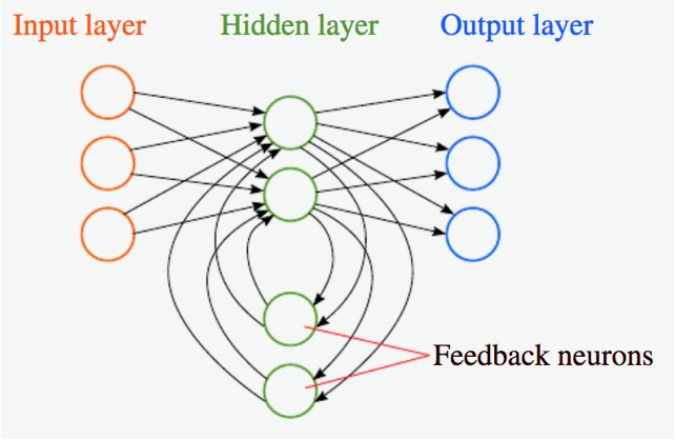

Fig. 3. Recurrent Neural Network Diagram

B. Model Two: Recurrent Neural Network on Sound Data

As described earlier, after extracting features the sound data had been transformed into a matrix representing the MFCCs at each time step. These features are then passed into a recurrent neural network (RNN) for classification. This network consists of one hidden long short-term memory (LSTM) layer with 400 output units, which are then passed to a softmax layer with four output units for classification. The loss function used is the multi-class version of cross-entropy loss, called "categorical cross-entropy loss."

$$CE(y, \hat{y}) = -\sum_{k=1}^{K} y_k \log(\hat{y}_k)$$

RNNs are a specific type of neural network in which activation values of units are used to inform other units. In this way, the hidden layer is able to feed back calculations at previous time steps (see Fig. 3) [7]. The cycle created by these units allows the RNN, unlike feedforward neural networks, to understand time-series information. This makes them ideal for use on frequency information that has time-based dependency (the MFC values at a given timestep t are not independent of the MFC values at $t-1$). However, historically RNNs have had problems with gradients either disappearing or exploding to extremely large values over long time intervals. The point of an LSTM layer is to remove this problem through the use of "forget gates," which control what information gets passed back to a unit. Each LSTM cell has a forget gate, and during backpropagation constant error flow is enforced such that the gradient will never vanish or explode. The end result is that the LSTM can learn information about dependencies and relationships that can span thousands of time steps [8].

C. Model Three: k-Nearest Neighbors on Image Data

After feature extraction, the pixel data from each image had been reduced to 3,200 features using PCA. Before attempting to classify the data, the features are compressed further using Linear Discriminant Analysis (LDA). LDA is a similar method to PCA in that it attempts to project a list of n-length feature vectors onto a lower-dimensional space of length $m < n$.

TABLE I
FINAL RESULTS

Model	Development Set	Test Set
Model One	0.8925	0.9050
Model Two	0.7525	0.7425
Model Three	0.6975	0.7175
Final (Weighted Sum)	0.8825	0.8525
Final (Linear Regression)	0.8925	0.9050
Final (Neural Network)	0.9075	0.9175

However, unlike PCA, LDA is supervised: where PCA attempts to maximize the total distance between each data point in the new m-length feature space, LDA instead attempts to maximize the between-class distance (called "scatter") while simultaneously minimizing the within-class scatter. LDA is thus a better method specifically for classification than PCA. After implementing LDA, each image is represented by only 20 features.

These features are then used to train a k-Nearest Neighbors model with $k = 5$. For each output to predict, k-Nearest Neighbors model takes in its RGB feature matrix, determines the 5 closest RGB matrices from the training set based on the L_2 norm of their differences, and predicts the song's genre based on a plurality vote of these 5 neighbors, taking the closeness of the neighbors into account in the case of a tie.

D. Final Model: Weighted Sum of Model Output

After building each of the models separately, the results are used to inform a final model. The vision behind this model is to combine the outputs of each of the intermediate models, and attempt to create a final prediction that is more accurate than any of the other models alone. Three different models are used to combine the intermediate outputs.

The first is simply a linear combination of the outputs of the final layers (probabilities for the RNN and Naive Bayes models, and the predicted classes for k-Nearest Neighbors, weighted by development set accuracy). Then, the predicted class is simply whichever class has the maximum combined value.

The second model is a basic linear regression model. The same probability outputs from above (although the k-Nearest Neighbors output was unweighted) are used as features to train and test the model.

Finally, the third model uses the same features as the linear regression model described above. However, this model is a neural network with one dense, five unit hidden layer using linear activation, followed by a softmax layer with four units used for classification.

V. RESULTS AND DISCUSSION

The results from each of the different models can be found in Table I. As can be seen, the intermediate model that performs the best on both the testing and development sets is the k-Nearest Neighbors model on the image data. This is surprising, considering the simplicity of the model compared to the RNN model. The RNN and the Naive Bayes model perform relatively well, but are not nearly as accurate. The confusion matrix for the k-NN model can be seen in Fig. 4. Interestingly, all of the Metal songs and nearly all of the Christian songs were classified correctly. However, many of the country songs were misclassified as Metal.

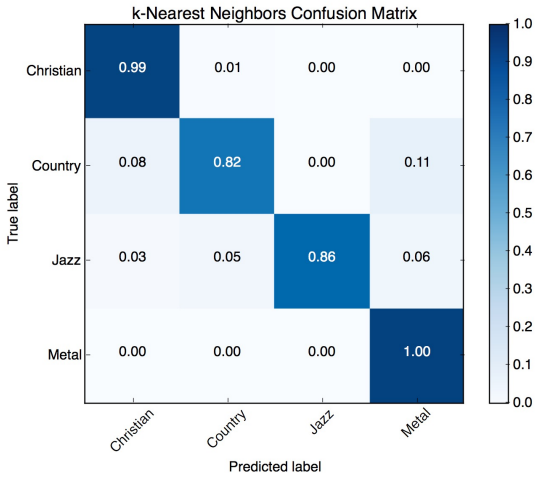

Fig. 4. k-NN Confusion Matrix

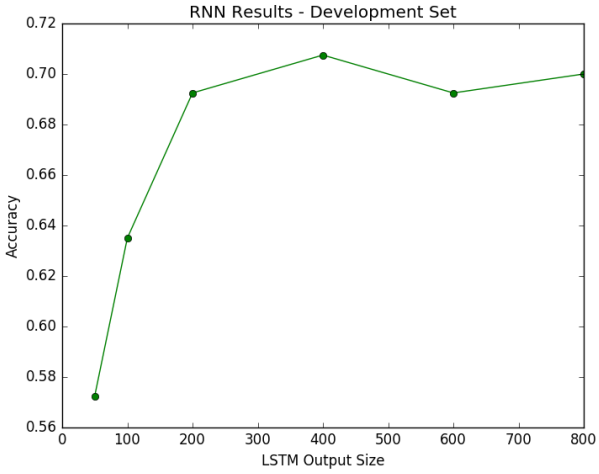

Fig. 5. RNN Accuracy

When creating the RNN model, we considered using more complex neural networks. Other work has shown that stacked LSTM layers often outperform single layers, as multiple LSTM layers allow for more sophisticated time-series information to be obtained. However, in our case a single-layer LSTM performs the best on the test set. We also tried different numbers of output units from the LSTM layer, and found that decreasing the number of units to 400 instead of keeping 800 (the same length as the input vector) improved the accuracy of the model, probably by allowing the RNN to learn more general song features. However, decreasing the number of output units any further made the model perform significantly worse. This relationship can be seen in Fig. 5.

For the final model, we initially tried a weighted sum approach, which works like a majority vote, but weights each

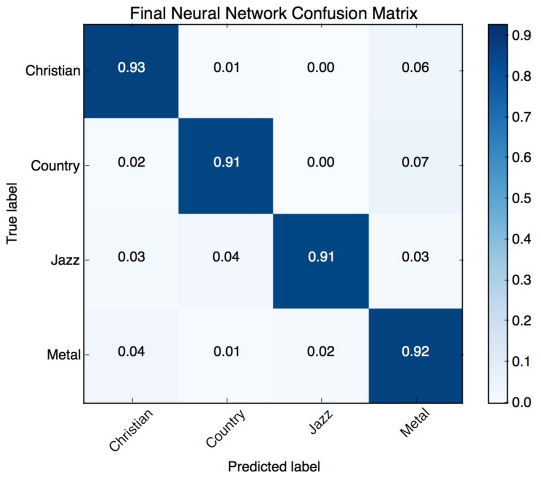

Fig. 6. Final Model (Neural Network) Confusion Matrix

vote based on the model's confidence, which are the softmax probabilities output from the intermediate models. However, as can be seen, the weighted sum does not perform as well as simply taking the output from the k-NN model. It is worth noting that when we use this approach on only the output from the RNN and Naive Bayes models, the final test accuracy is 0.7625, which is higher than either of the two model accuracies. Thus, we concluded that the idea of combining the model outputs is still sound, but the skew in the accuracy of the intermediate models prevents it from improving the overall accuracy.

Thus, a more complex model is necessary to fix this problem. The second final model performs linear regression on the outputs of the intermediate models. As can be seen, the output of the linear regression model matches that of the image data. However, it turns out that the linear regression has the exact same output as the k-NN model alone. In other words, since the image model is much more accurate than the RNN and NB models, the final linear model simply learns to use the k-NN model to classify new data, without taking into account the features from the other models. We thus decided to again use a more sophisticated model.

The final neural network model is the only of the three that manages higher development and test set accuracy than the intermediate models. The confusion matrix for the test set is in Fig. 6. Compared to simply using the image data, the final combined model has much more standard accuracies across the four categories. This suggests that the model is more generalizable than the k-NN model alone.

VI. Conclusion

Perhaps the biggest takeaway from this project is that there is a huge possibility for more exploration of album artwork in regards to genre and other information about the song. Given that our project covers a wider breadth of features, a more detailed and focused look at the artwork is merited.

In addition, it is valuable to show that models using features as disparate as artwork, waveforms, and lyrics can be combined to produce a model that is more accurate than any of the three alone (as was done with the final neural network model).

Moving forward, we would perform more rigorous cleaning on the lyrics and use more sophisticated NLP models on the lyrics to achieve better performance with our Naive Bayes model. We would also include several other genres, as a majority of Spotify songs do not fall under any of our chosen genres. Finally, we could also look for other features to add, such as metadata about the song like beats per minute or song title.

References

[1] N. Patch, *Meet the man classifying every genre of music on Spotify*, thestar.com. 2016.
[2] M. Haggablade, Y. Hong, K. Kao, *Music Genre Classification*. 2011.
[3] B. Logan, *Mel Frequency Cepstral Coefficients for Music Modeling*, Transactions of the International Society of Music Information Retrieval. Ubiquity Press, 2000.
[4] R. Mayer, R. Neumayer, A. Rauber, *Rhyme and Style Features for Musical Genre Classification by Song Lyrics*, Transactions of the International Society of Music Information Retrieval. Ubiquity Press, 2008.
[5] S. Kuznetsov, *55000+ Song Lyrics*, Kaggle. 2016.
[6] *Spectral Density*, Wikipedia.
[7] M. Galetska, *Intelligent Predictions: an empirical study of the Cortical Learning Algorithm*, Hochschule Mannheim. 2014.
[8] S. Hochreiter, J. Schmidhuber, *Long Short-Term Memory*, Neural Computation. 1997.