Music Genre Classifier

Tannavee Kumar, Nick Stapleton, Matthew Marlow, Jose Torres,
Cameron Fitzpatrck, Kevin Lee, Chance Stewart, Jatin Mohanty,
Jiahui Dai, Spencer Grossarth, Luc Nglankong

Abstract

This paper addresses the ongoing work in the field of machine learning classification, and specifically music genre classification, in which a music genre classifier takes a song as an input and returns the genre that the song is most similar to as the output. The chosen approach examines single class classification on 16 possible musical genres through a variety of different machine learning models and algorithms to see which method results in the best accuracy. The algorithms and models built were Logistic Regression, Support Vector Machine (SVM), and a Feed Forward Artificial Neural Network (FFANN). To further optimize the FFANN, a parameter sweep was done to find the optimal FFANN. Eventually this resulted in a FFANN with one hidden layer and (n+1) nodes per layers to categorize songs based on features such as mel-frequency cepstrum coeeffiecients (MFCC) and spectral density. On average, the FFANN outperformed the SVM and logistic regression classifiers when using ReLu internal activation functions and soft-max output activation functions. In addition, this paper proposes a rank metric that can be used to determine the performance of a classifier and provides a website interface to visualize the performance of the model.

https://github.com/nhstaple/ecs-171-music-genre-classifier/archive/v2.0.zip
https://github.com/nhstaple/ecs-171-music-genre-classifier

1 Introduction

Supervised classification is a common problem that many machine learning algorithms are created to solve. The main goal of supervised learning, particularly amongst classifiers, is to build a model that takes in a sample's features as inputs and returns the sample's classification [1]. So far, there has been "extensive work" done within supervised music genre classification. The common method amongst this field is to take "taxonomy" of genres, and " try to map a database of songs into it by machine learning algorithms" [2].

However, a recurring issue being faced in genre classification is the discrepancy in accuracy when applying datasets with different distributions of classes (genres) [2]. In other words, datasets with imbalanced distributions have lower accuracy rates. To address this issue, researchers have attempted to normalize their datasets, although normalization in this instance is nontraditional. Datasets are adjusted so that each genre has the same number of samples. While this may not decrease the overall number of classes, it doest make the data more balanced by reducing bias [2]. Unfortunately, this often still hinders the accuracy since creating a balanced dataset typically leads to a smaller dataset overall [2]. We noticed that while studies examined accuracy rates, they have yet to show how close the predictions were to the actual genres. Instead of the typical categorical cross-entropy accuracy, we were interested in knowing how far off the predictions were in a standardized manner by comparing the probabilities of a sample belonging to various genres and ranking them. We believe that this is an important metric since accuracy paints a very black and white picture; however, music is more complicated than that. Even if predictions are a little off, learning how close it was can shed light on similar features amongst genres.

This paper contributes a rank system through a visual interface in order to communicate how accurate our predictions are through the best machine learning learning model that we tested. Furthermore, for predictions that were inaccurate, the interface hopes to bridge the gap by indicating the rank of the actual genre in the predicted genre vector. If the predicted genre is very close to the actual then that could mean the genres have similar feature values.

2 Methods

2.1 Data and Machine Learning

Before we began building our models, we examined our options. Since we had a large data that was already labeled, choices we considered were Logistic Regression, Support Vector Machines (SVM), Decisions Tree Algorithms, Random Forests, K-Nearest Neighbors, and a Feed Forward Artificial Neural Networks (FFANN) [3]. Given the plethora of features that were present in our dataset, it was apparent that utilizing a Decision Tree would yield overfitting [3]. Further research also pointed out that Random Forests could pose problems due to "lack of diversity" in our dataset due to imbalanced number of samples in each genre [4]. While K-Nearest Neighbors seemed like a viable option at first, after discovering that they are limited to only being able to provide predictions based off of "memorizing the dataset," it was clear that it would not be a great fit for a task such a music genre classification due to its diverse nature [3, 5]. Thus, we decided to proceed with Logistic Regression, SVM, and FFANN.

2.1.1 Data Management

The labeled data used in this project was obtained from the Free Music Archive(FMA) [6]. There were four csv files within the dataset, tracks.csv had track metadata, genre.csv had all the genres and hierarchical information, and features.csv contained audio features for all the tracks (samples). The features consisted of several audio attributes such as Chroma, Tonnetz, MFCC, Spectral Centroid, Spectral Bandwidth, Spectral Contrast, Spectral Rollof, RMS Energy, and Zero-Crossing Rate. Statistics such as the mean, medium, minimum, etc. were calculated over a window of the song, where each window is 2048 samples of the song, for each of these main features. Therefore, the entire feature data frame consisted of 518 columns or features. For a visual representation of the data, refer to the "dataOverview.md" found in this project's repository. The fourth file, "echonest.csv," contained audio features that were obtained by a company called Echonest. Due to lack of documentation describing the features or how they were gathered due to their propriety nature, they were not considered in this project.

The dataset's samples are labelled according to 161 genres. These genres are organized according to a built in hierarchy. Out of the 161 genres, 16 are what we refer to as root genres. Each root genre consists of many sub-genres which make up the rest of the genres. When doing genre classification we attempt to predict the root genre of a given sample (track).
created three datasets. The small and medium datasets were optimal for prototyping the code used in analyzing which data structures, features, and methods provide the best music genre prediction results. The small dataset was composed of uniform samples from 8 of the 16 total top genres, and was used to test the functionality of our programming. The medium dataset had song samples from all sixteen genres but did not have a uniform distribution of results. The large dataset is similar except it is possible for songs to be labeled more than one genre.

2.1.1.a Preprocessing and Data Organization While it is standard to detect and remove outliers, particularly when utilizing a large dataset, we decided not to do so for a few key reasons. Although certain music may belong to the same genre, they can still vary drastically; therefore not removing outliers will allow the samples resemble authenticity in variation. Furthermore, the authors of the paper that accompanied the dataset point out that "it is hard to set a threshold" when determining whether a song is an outlier or not [7]. Even more, determining what makes a song an outlier is difficult to determine through quantitative means since genre classification can be subjective [8]. Finally, the authors mention that even if outliers are left "the small number of outliers will not impact performance much anyway" [7].

To assist music analysis, the provided dataset was divided into three subsets: large, medium, and small subset. The small subset has 8,000 samples with the 8 most popular genres. This subset is balanced as each genre contains exactly 1000 tracks. The medium subset consists of 25,000 tracks. Each track in this set contains exactly 1 out 16 root genres. This set is unbalanced with 21 - 7,103 tracks per root genre. Finally, the large subset contains the entire data set which is also unbalanced with 1 to 38,154 tracks per genre. The entire subset consists of 106,574 tracks (samples); however, only 47 percent of the tracks had a root genre. After dropping the tracks that did not have a label we were left with a new subset that we called 'cleanLarge' which consists of 49,598 tracks.

2.1.1.b Feature Selection To ensure high accuracy amongst the models, feature selection algorithms were used. This would also help train models faster and reduce overfitting. The two feature selection algorithms used were univariate feature selection using python sklearn library's SelectKBest with the "f classif" scoring method, and minimum Redundancy Maximum Relevance Feature Selection, or mRMR [9].

The process of univariate feature selection involves a straightforward comparison of the scores of all features in the feature set, using any scoring function [10]. Our feature selection used the f classif function, which is a function that applies the ANOVA F test over the data. This function was used because it worked well for our dataset, unlike other scoring functions which are not tailored for classification problems. Some scoring functions have certain restrictions that our dataset did not meet, such as the chi-squared test, which requires that the dataset has no negative values. The ANOVA F test scoring method clusters each feature's data by the classifications they belong to [11], and then compares the means of all of them to see how far apart the mean value for each classification are. A variance is computed for all of these means, and if the variance is higher, then the values are more distinctly split into the different classifications for that feature, and the feature will have a higher score. If the variance is low, then the different classes are not easy to distinguish from one another using that feature, and it will have a lower score.

The second method of feature selection we tried was mRMR [12]. The idea behind the minimum Redundancy Maximum Relevance Feature Selection is exactly what the name implies: to find the features with the maximum level of relevance to the classes that they belong to, while having the minimum amount of redundancy between selected features. This is done through creating a scoring function for both relevance and redundancy, and setting each feature's overall score to the difference of relevance minus redundancy. A more detailed explanation of how the scores for relevance and redundancy are calculated can be found in the academic literature for mRMR [13].

After using univariate selection, we noticed that the majority of the features selected were from the mfcc and spectral contrast categories, so we decided to add those categories to our comparison for feature selection methods. We compared mfcc, spectral contrast, our full univariate selection, and mRMR, and found that the combination of mfcc and spectral contrast achieved the highest accuracy. Our final model used the features of mfcc and spectral contrast that we got from observing the trend of the features returned by univariate selection.

2.1.1.c Proper setup for website building Other areas of the data management were related to making our website deployable. We made scripts to pickle the large csv files into more manageable sizes to circumvent memory limits and speed limits of a standard csv file. We compressed the "features.csv" file from 951mb to a 443mb pickle and "echonest.csv" from 44mb to a 26.3mb pickle. Additionally, accessing pickles offers faster response times when opening them compared to csv files.

We created an interface for the machine learning team to have access to the data from our features and tracks dataset. This script converts csv files to pkl files(binary file), getting features, subsets of given data frames, getting random songs to test, and returning bins of songs corresponding to their respective top genre. Additionally a K fold validation function was placed as well to aid training for the machine learning team.

Furthermore, there was a need to implement a database so the back end server could access the necessary data without opening files every time which was time consuming. We began by attempting to implement a SQLite database. However, when querying this database, the operations were much too slow. For the sake of saving time the data management team decided to make a "pseudo" database consisting of instances of dataframes which the back end could query with a much faster response time. This allowed for faster response times on our website.

2.1.2 Building Models

To set a benchmark for our prediction, the prediction vector of a song is randomly assigned to one of the sixteen genres uniformly via the Naive Method.

2.1.2.a Logistic Regression Scikit-learn's logistic regression model was applied to predict genres from the small data subset. Logistic regression inherently is a binary classifier; given a set of features in a sample, it predicts whether the sample belongs to class 0 or 1. Logistic regression is based upon the sigmoid function:

$$g(x;w) = sigm(w^Tx) = \frac{1}{1+e^{-w^Tx}} = \begin{cases} 0 & \text{if } g(w^Tx^{(i)}) < 0 \\ 1 & \text{if } g(w^Tx^{(i)}) \geq x \end{cases}$$

Then the probability of a class is:

$$p(y^{(i)}|x^{(i)};w) = g(x^{(i)};w)^{y^{(i)}}(1-g(x^{(i)};w))^{1-y^{(i)}}$$

The objective is to maximize the log likelihood given M samples:

$$\text{argmax}(w)\Sigma_{i=1}^{M}(y^{(i)}\log g(x^{(i)};w)+(1-y^{(i)})\log(1-g(x^{(i)};w)))$$

To perform multiclass classification, a one-vs-rest scheme was used [14]. This method takes one class as positive and the rest as negative, and trains 8 classifiers for the 8 genres in the small subset of data. This is less computationally expensive than a one-vs-one approach, allowing for faster training and the ability to use more samples during the initial fitting, which produced better results [14]. The model was fit four separate times using the features: "mfcc", "spectral contrast", "mfcc and spectral contrast", and the top 200 features from mRMR analysis in each construction.

2.1.2.b Support Vector Machine The original paper that analyzed the dataset used an SVM to classify. The SVM was built from Scikit-learn's linear SVM model, which supports a linear kernel. An SVM is a specific case of a margin classifier. Given M samples with N features and two classes, an SVM draws a hyperplane in the feature space to separate the two classes [15]. The criterion is that this decision surface is maximally far from any data point.For instance, given that the two classes are + and -, the classification function for some sample x is:

$$f(x)=\text{sign}(w^Tx+b)$$

where sign() is a function that extracts the sign (+ or -) of a number, w is the weight and b is a bias term. We can rearrange this as:

$$y^{(i)}(w^Tx^{(i)}+b)\geq 1$$

The objective is to then find a w and b such that:

1. w^Tw
2. $\forall(x_i,y_i)(y_i(w^Tx_i+b)\geq 1)$

Note that because it only extracts the sign (+1 or -1) of a sample, it is a binary classifier. The SVM is used to predict genres from the small subset of data. To perform multi-class classification, a one-vs-one scheme was used. This method considers each binary pair of classes (e.g. pop and rock) and trains on the subset of data containing the binary classes. Thus, it trains 28 classifiers for the 8 genres in the small subset of data. Although this is computationally expensive compared to a one-vs-rest approach, this method produced a higher accuracy than an ovo method for the SVM. The model was fit four separate times using the same features as the logistic regression.

2.1.2.c Feed Forward Artificial Neural Network In accordance with our hypothesis, we constructed a Feed Forward Neural Network (FFNN) to solve our problem of multi-class classification. As with most FFNN's geared toward multi-class classification, we chose the softmax function to act as the activation function in the output layer. This formatted the output of the network in a way that was easy to analyze: a 16 dimensional vector where each entry represents the probability of the input song being a member of that particular genre. In the hidden layers, we chose to use a rectified linear unit (ReLU) activation function. Between the typical choices of sigmoid, hyperbolic tangent, and ReLU, ReLU produced the best results. For the optimizer in all nodes, we chose to use the Adam algorithm due to the much faster convergence than seen in the classical stochastic gradient descent (SGD).

Using the softmax activation function in the output layer produces a vector of probabilities as the output. In order to evaluate this output, we considered the highest probability genre to be the model's prediction. For example, if the model predicted "Rock" with 30 percent probability, "International " with 28 percent probability, and "Pop" with 25 percent probability for a song was labeled "Rock, Blues", it would be considered correct since "Rock" is included in the list "Rock, Blues."

In the process of constructing the FFNN, a parameter sweep was conducted in the form of a grid search to determine the hyperparameters that give the best results. The hyperparameters that we modulated were the number of hidden layers in the network and the number of nodes per hidden layer which we kept consistent for all hidden layers. The search was conducted by creating a model with each combination of number of hidden layers and number of nodes per layer. Each model was tested by predicting the genres of 500 songs from our validation set. The accuracy, error, and mean score achieved by each model were recorded. In Figure 2.2 below, the results of the training and testing accuracies, the error, and in Figure 2.2, the score of each combination are shown.

2.1.3 Gathering Performance

Determining performance across the different methods required different approaches.

2.1.3.a Logistic Regression The model was fit using 7000 random samples. We discovered that this is not feasible using an ovo approach because training takes too much time. Training on fewer samples with an ovr method resulted in worse testing accuracy. Accuracy was calculated by predicting the genres of 1000 test songs that the mode had not seen before. The genre with the highest probability was taken to be the genre that the model predicted. Predicted genres were then compared to the ground truth genres, and the percentage of correct predictions was accuracy. Results can be seen in Figure 1.

2.1.3.b Support Vector Machine The model was created to fit 1500 random samples. Model fitting was not done on 7000 samples like in logistic regression because it was too computationally expensive to use an ovo classification. However, compared to an ovr method fit on more samples, the ovo SVM had a much higher accuracy. The testing set consisted of the 6500 samples the model was not fit on. Accuracy was calculated by predicting the genres of the test set, then comparing the genres to the ground truth genres. Though the SVM was fit on comparatively few data points, Figure 1 shows that the accuracy of the SVM was not far behind the other models.

2.1.3.c Feed Forward Artificial Neural Network Our group created the metric "rank" that was used to score each prediction. The neural network produces a vector of predictions sorted from most likely to least likely. The rank of a prediction is the location in the prediction vector that the actual category is contained. A rank of 1 corresponds to a perfect prediction while a rank of 16 corresponds to the worst possible result. For example, if the model predicted "Rock" with 30 percent probability, "International" with 28 percent probability, and "Pop" with 25 percent probability for a song was labeled "Pop", then the rank would be three since "Pop" is the third most likely prediction.

2.1.3.d Detecting Best Model From Figure 2.2 the best result produced from the parameter sweep (4 layers and 32) had an average rank of 1.8 on 500 songs. When extended to 1000 songs the average rank for all the predictions was 1.946. However, a better result was achieved when advised by the universal approximation theorem using n + 1 nodes per hidden layer [16]. For one hidden layer the average rank for 1000 songs was 1.792, which outperforms the best result from the parameter sweep despite being an overfitted model. Increasing the number of layers improves the average rank but also increasing the degree of overfitting for the model.

2.2 Engineering

2.2.1 UIUX

The overall design of site it meant to be straight forward with a search bar to input a song title. There is also a "Feeling Lucky" button that can generate a search for a random song. The second webpage

2.2.1.a Building the Foundation The front-end foundation was built using HTML, CSS, and JavaScript. We used React as a framework for JavaScript and npm for development and testing. We used Vanilla CSS for styling. There is only one HTML page, yet there are multiple page views. This is accomplished by using HTML as an entry point for JavaScript insertion. React creates a virtual DOM, and manipulates the DOM elements by setting states. Upon a setState() function call, React will re-render the DOM.

To fully implement the front end and use React, we opted to code using the syntax friendly jsx. However the browser does not recognize jsx, so we had to use Babel as a compiler to convert jsx to js. The browser can recognize and run js files. In order to run Babel as a compilation tool, Gulp was used. In a gulp file, we are able to set a task. That task is conveniently named "babel" and it converts all jsx files to js in the src folder.

In order to send information to the backend we used a HTTP request, in the form of AJAX. The AJAX request sends information to the back-end (explained below) and awaits a response. It receives a JSON for a response and then parses that JSON into an object. Once we have an object, we can easily access fields of that object and use JavaScript functions to access the data and display it on the browser.

2.2.1.b Polishing and Incorporating Design Elements The final design of the UI features a simple search input field with two buttons. The "Search" button will send an AJAX request to the backend containing the song name entered in the input field. The "Feeling Lucky" button sends an AJAX request to the backend with a random flag and no song title, which will trigger the backend to search for a random song in our song database. The second page of the website shows the results of the genre prediction. The results page features the song title, artist, predicted genre and the determined probability of the predicted genre. To demonstrate how effective our neural network is, we also show the actual genre and our determined probability of the actual genre. The song rank and model rank are also shown. For reference, we integrated a YouTube search link at the bottom of the page to help find the actual song and to determine if the predicted genre is plausible.

2.2.2 Software Engineering

The general job of the backend was to build the infrastructure to connect all the components of the project.

2.2.2.a Framework The back-end framework is a python flask server, which is able to receive front end website request and response processed song result as a json object. Because we host our frontend and back-end under the same domain – localhost, CORS problem occurs. In order to avoid the problem, we configure the proxy for our front end API. The front-end framework is React for JavaScript. React uses a virtual DOM to create dynamic webpages using a single HTML file. React uses states and props to implement this idea, and we did not use redux as a state management system.

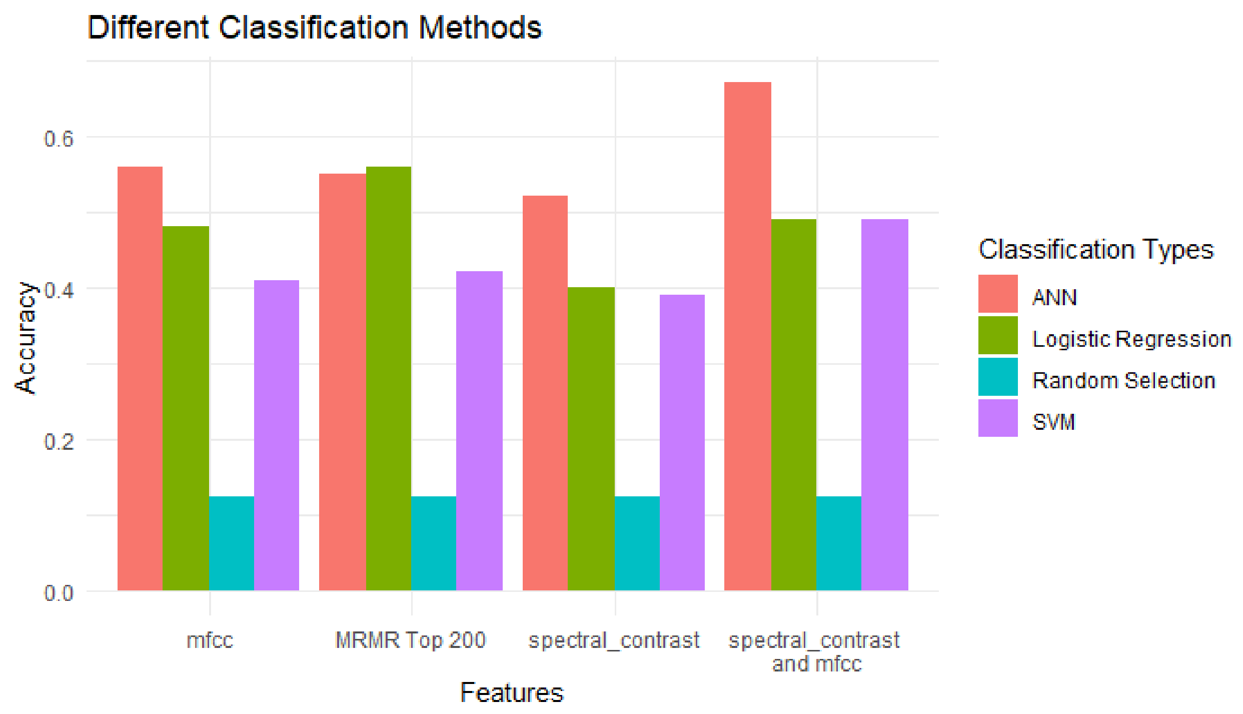
2.2.2.b Interface The interface for the back end is defined in $Back_End/song_result_interface.py$. This is a dictionary object that mimics an asynchronous JSON response object with fields for each prediction. When a song is queried from the database an object with fields specified in $song_result_interface.py$ is returned. Then the query result is modified in the back end pipeline (see section 2.4) before being sent to the front end. The database query, backend pipeline, and AJAX response all use the same interface.

2.2.2.c Deployment The website is hosted on Amazon Web Services. The website is deployed using Amazon's Elastic Beanstalk. While it is possible to directly deploy Python applications to a Python specific platform, a more general platform was needed for the use of React in addition to Python. To create an environment that would support React and Python, Docker was used to containerize all of the code. The base image used was the official Tensorflow image. Because all of the music data is stored on pickle files and loaded directly into a dataframe, the application uses a large amount of memory. To handle the high memory use, the standard T2.micro instance was not enough to support the backend with only 1 GB of memory. Due to this large memory requirement a T2.small instance, with 2 GB of memory, was configured and used.

2.2.2.d Pipeline There are 4 major components of this website: the frontend, backend, data management, and machine learning. The backend pipelines all of these components together to make a complete application. The backend first initializes the neural network and an instance of our database before the pipeline begins. In the pipeline, the backend first collects an AJAX request from the frontend containing a song name. Next, the data management interface is called to query the song data. If there are multiple songs that matches the given song name, they will all be returned to the pipeline. Then, the machine learning's interface is called to produce prediction data for the song(s). Finally, the prediction data is sent to the front end in a JSON response object. This pipeline is capable of handling multiple song requests using the same neural network.

3 Results

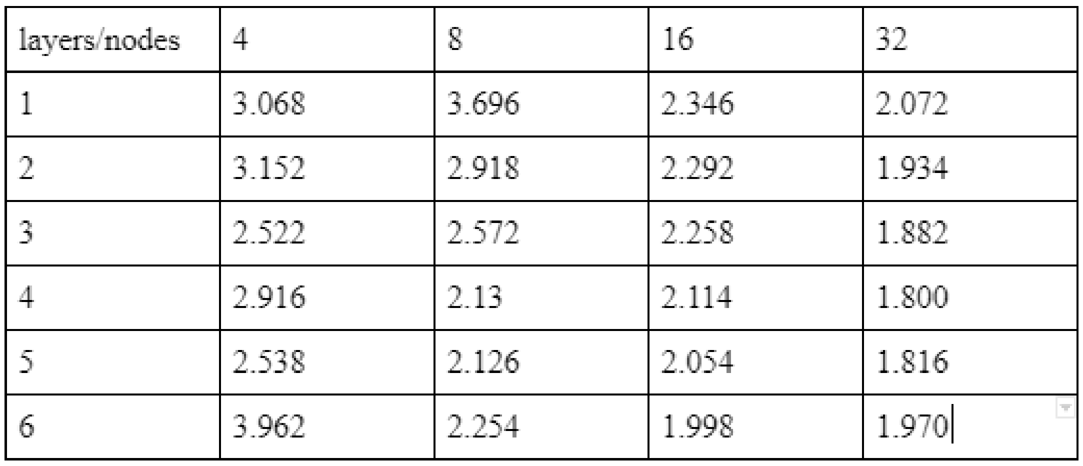
3.1 Figure 1: Table comparing accuracy of LR, SVM, ANN



All classification methods outperform the naive classifier that randomly assigns a uniform prediction. The final feed forward artificial network outperforms other classifier methods on most combination of features. The network had a significant increase in performance when combining the two features strongly identified by univariate selection. The hyperparameters for the network were inspired by the universal approximation theorem for continuous functions, using ReLU activation functions and a node depth of $n + 1$.

3.2 Figure 2: Parameter Sweep Results

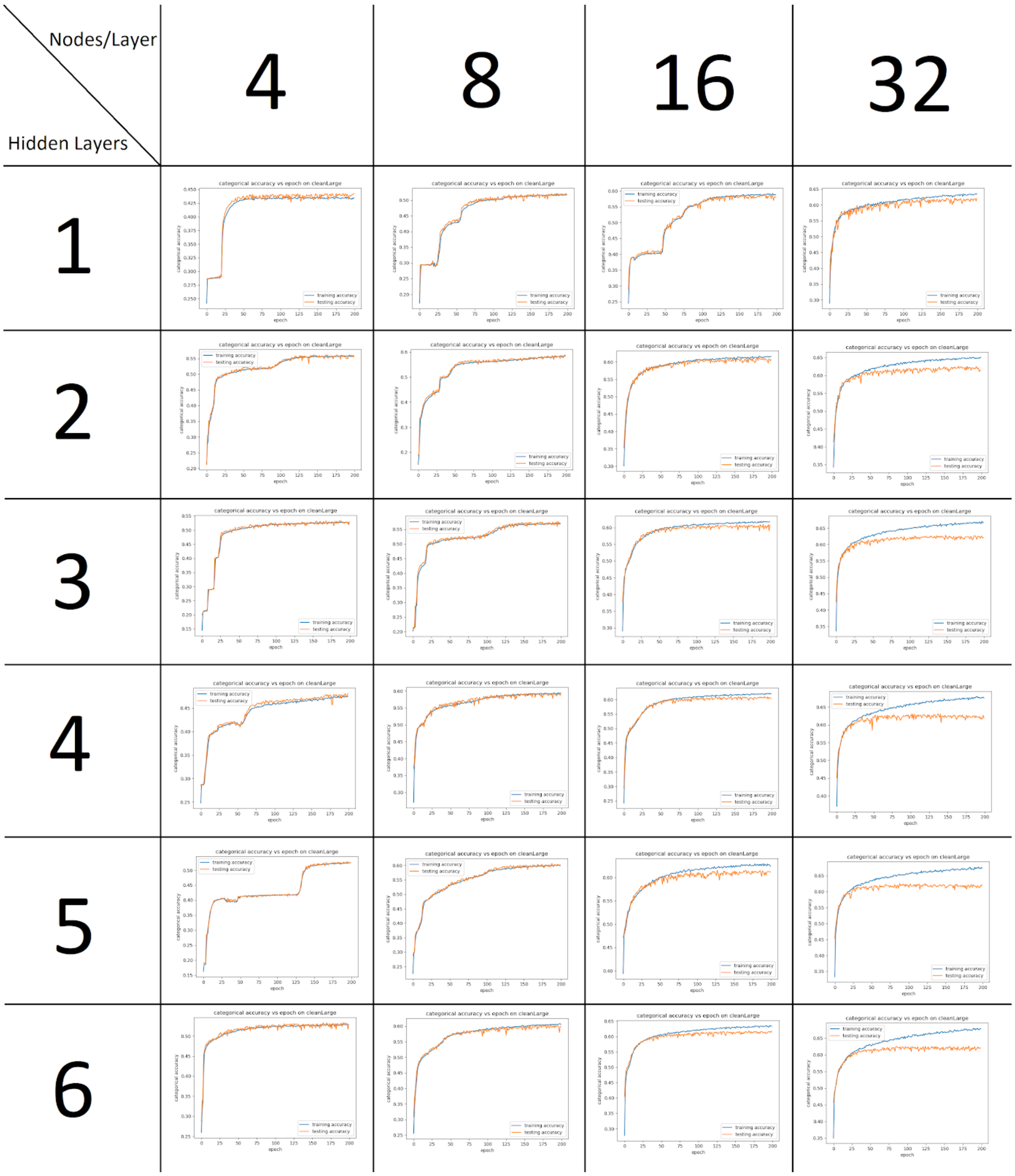
3.2.1 Figure 2.1: Mean Score For Parameter Sweep on Large Dataset



layers/nodes	4	8	16	32
1	3.068	3.696	2.346	2.072
2	3.152	2.918	2.292	1.934
3	2.522	2.572	2.258	1.882
4	2.916	2.13	2.114	1.800
5	2.538	2.126	2.054	1.816
6	3.962	2.254	1.998	1.970

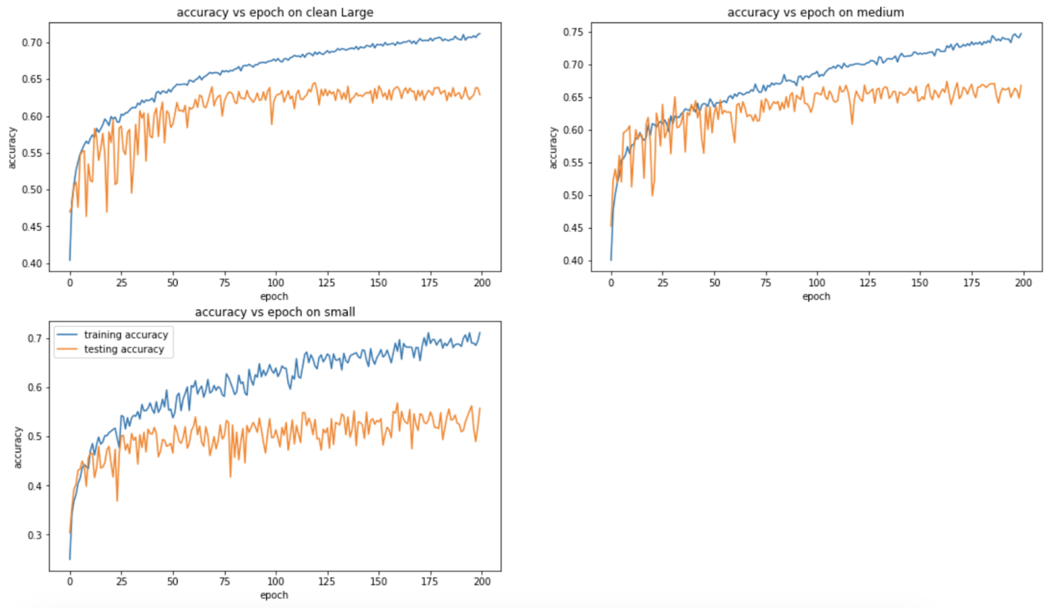
According to the scoring metric that we created (0 being correct, and 16 being worst), the FFANN with 4 hidden layers, and 32 nodes per layer performed the best on average with a score of 1.800.

3.2.2 Figure 2.2: Training and Testing Accuracy For Parameter Sweep on Large Dataset



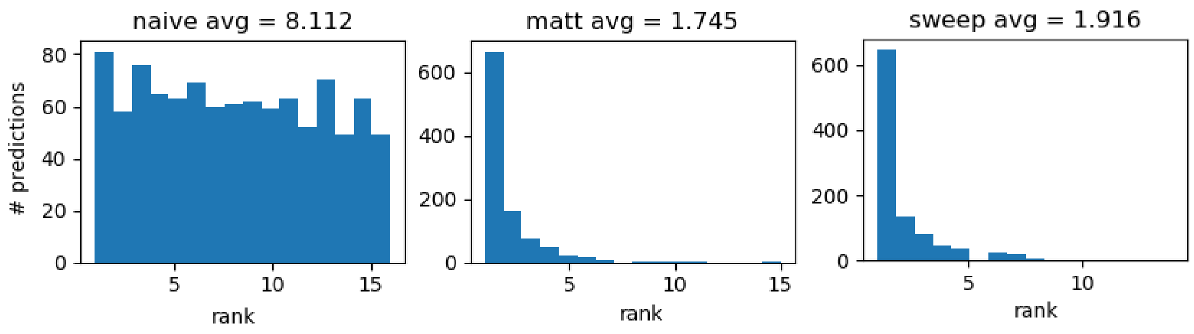
In the parameter sweep, the FFANN with 4 hidden layers, and 32 nodes per layer performed the best. However, the results show that there was overfitting since the training accuracy (blue) was so much higher than the testing accuracy (orange).

3.3 Figure 3: Training/Testing Accuracy for Best Performing FFANN (1 hidden layer, n + 1 nodes/layer)



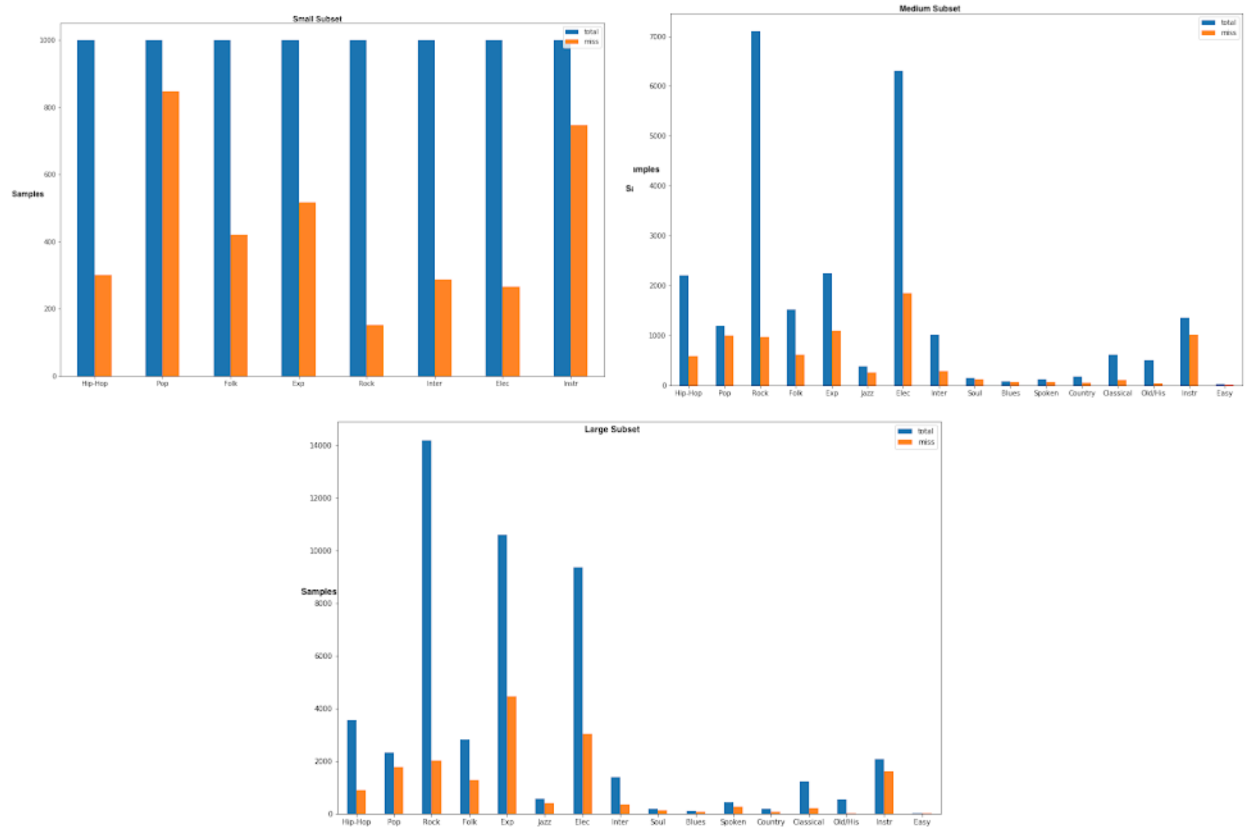
This figure shows the final model training set and test set accuracy vs epoch plots for the small, medium, and large datasets. The blue curve represents the final model training set accuracy and the orange curve represents the final model test set accuracy. The training set reported a greater accuracy than the test set for all three plots, and the medium dataset resulted in a generally slightly higher accuracy compared to the other datasets. The small dataset had the highest variation in training accuracy.

3.4 Figure 4: Training/Testing Accuracy for Best Performing FFANN (1 layer, n + 1 nodes)



Comparison between the ranking distributions a naive model that randomly guesses the genres in different orders, our final model (called "matt"), and a model that we obtained from our parameter sweep. The random model has a roughly even distribution of scores, while the other two models are skewed towards scores closer to 1. The final model has a higher number of scores close to 1, leading to a lower average score, although it has a higher number of far outliers compared to the optimal parameter sweep model.

3.5 Figure 5: Performance of Best Performing FFANN per Genre for Small, Medium, Large Dataset Size



The figure from left, right, bottom respectively shows accuracy in small, medium, and large datasets per genre. Blue bar represents total samples while the orange bar represents the number of misclassifications. Pop and instrumental performed worst in small, with Rock being the best. Electronic did the worst in medium, with Easy doing the best. In large, Easy continued being the best, with Experimental being the worst.

4 Discussion

One hypothesis that we had in regards why a FFANN performed better than an SVM and logistic regression in Figure 1 was due to the number of features we had, and the size of our dataset. The results of the parameter sweep show that the more nodes we used, the better the achieved accuracy and lower the mean score. However, it should be noted that these models typically also produced models that were over-fit to the training data. This can be seen in Figure 2.1 where the training accuracy diverges from the testing accuracy by a decent margin. However, despite overfitting, the mean scores achieved by the higher node count models on the validation set were indeed lower(better) and for this reason, we overlooked the slight overfitting of the model in favor of the improved generalization capacity of the model. An attempt at using an n + 1 node network produced better results than the optimal parameter sweep configuration. It was interesting that the small dataset had the highest variation of testing accuracy with 8 classes. The n + 1 model was deemed better than the best model gotten from the parameter sweep due to better accuracy; however, it was less stable. We believe that training the model with 200 epochs rather an 100 was the major cause of overfitting for this model as well.

From Figure 4, the distribution of ranks for the naive predictor is roughly uniform as expected. The distribution for the optimal parameter sweep model performed marginally worse than the n +1 node network. This could be do to overfitting of the n + 1 width network meaning that the model is not general. In Figure 5 On the evenly distributed small dataset pop, instrumental, and experimental were the most misclassified genres. The medium and large datasets were not uniform and had a bias towards rock, experimental, and electronic samples. Pop and instrumental were among the most misclassified genres across datasets for the n + 1 node network. The reason for this could be mislabeled songs in the database from the free music archive creating a false negative. For example, “Records” by R Stevie Moore is predicted to be “Experimental” but the database has it labeled as “Rock.” R Stevie Moore is known for his experimental music and this particular recording is more experimental than it is rock n roll.

Our model was trained on an unbalanced data set, since the “normalized,” small dataset had too few samples for reliable prediction. This is why we used the large since it had all 16 with a large sample size. This means our predictions are skewed towards the representation of the large dataset. For the model to produce predictions that are general to music, then a balanced large dataset is required for all 16 genres. Even though we had to face this hurdle, we were able to prove our hypothesis correct, a FFANN would outperform a SVM and logistic regression algorithm. We believe that the rank system that we created from the FFANN to communicate how close the predicted genre was to the actual genre can shed light on similarity of features across genres, but also show how classifying music genre is very complex.

Future work can be done to allow predictions of songs from outside of the free music archive dataset. This will increase the amount of data the model can be trained on exponentially. Further exploration of optimal parameters combined with the extra FMA archive samples could improve generalized performance on all music (from 16 genres.). Even more, we hope to expand into the more complex realm of multi-class classification in music genre classification if successful. increasing the amount of songs that the model can be trained on would enable the creation of a balanced large dataset. The importance of a balanced dataset means that no genre of music is more or less represented. Then the model will be a generalize music genre classifier with an output dimension of 16 predictions. A balanced data set will better equip us with real-world issues like complex music that requires multi-class classification. The web interface developed by the UI/UX and Software Engineering teams would need to add the ability to send a raw sound file to the classifier. This would provide the sound processing and machine learning communities with an intuitive resource that requires no installing of code to receive a result to compare methods. To begin to work towards the road ahead, the front end would send a raw sound file to the back end. We would have to implement a custom waveform processing pipeline stage written in python, using a package like librosa. A data acquisition stage would need to extract the artist, title, top genre, and waveform of a large uniform set of songs. This can be done with music done in the public domain with the genre categories restricted to the set of 16 genres identified by an unsupervised clustering method, for example k-means. A fraction of the generated labels will need to be verified by manually to verify that the dataset has false negatives. Even more, our rank system can serve as an intermediary until multi-class classification in music genre classification is successful

References

[1] Kotsiantis, S. B., et al. “Machine Learning: a Review of Classification and Combining Techniques.” *Artificial Intelligence Review,* vol. 26, no. 3, 10 Nov. 2007, pp. 159–190., doi:10.1007/s10462-007-9052-3.

[2] vol. 23, no. 2, 24 Apr. 2006, pp. 133–141., doi:10.1109/msp.2006.1598089. *IEEE Signal Processing Magazine,* Addison-Wesley, Reading, Massachusetts, 1993.

[3] Roman, Victor. “Supervised Learning: Basics of Classification and Main Algorithms.” *Medium,* Towards Data Science, 31 Mar. 2019, towardsdatascience.com/supervised-learning-basics-of-classification-and-main-algorithms-c16b06806cd3.

[4] Tang, Cheng, et al. “When Do Random Research Forests Fail.” *When Do Random Research Forests Fail,* Towards Data Science, 31 Mar. 2019, towardsdatascience.com/supervised-learning-basics-of-classification-and-main-algorithms-c16b06806cd3.

[5] Zhang, Min-Ling, and Zhi-Hua Zhou. “ML-KNN: A Lazy Learning Approach to Multi-Label Learning.” *Pattern Recognition,*vol. 40, no. 7, July 2007, pp. 2038–2048., doi:10.1016/j.patcog.2006.12.019.

[6] Defferrard, Michaël, et al. “Mdeff/Fma.” *GitHub,* Cornell University, 15 May 2019, github.com/mdeff/fma.

[7] Benzi, Kirell, et al. “FMA: A Dataset For Music Analysis.” *ArXiv.org,* Cornell University, 5 Sept. 2017, arxiv.org/abs/1612.01840.

[8] Seyerlehner, Klaus, et al. “A Comparison of Human, Automatic and Collaborative Music Genre Classification and User Centric Evaluation of Genre Classification Systems.” *Lecture Notes in Computer Science Adaptive Multimedia Retrieval. Context, Exploration, and Fusion,* 2011, pp. 118–131., doi:10.1007/978-3-642-27169-4-9.

[9] Bisong, Ekaba. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners.* Apress, 2019.

[10] “1.13. Feature Selection¶ *Scikit,* Scikit-Learn Developers, scikit-learn.org/stable/modules/feature-selection.htmlunivariate-feature-selection.

[11] Albon, Chris. “ANOVA F-Value For Feature Selection.” *Chris Albon,* Chris Albon, 20 Dec. 2017, chrisalbon.com/machine-learning/feature-selection/anova-f-value-for-feature-selection/.

[12] “MRMR (Minimum Redundancy Maximum Relevance Feature Selection).” *MRMR Feature Selection Site,* home.penglab.com/proj/mRMR/.

[13] Peng, Hanchuan, et al. “Feature Selection Based on Mutual Information Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy.” *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 27, no. 8, Aug. 2005, pp. 1226–1238., doi:10.1109/tpami.2005.159.

[14] Fudong, Zhao, et al. “Comparative Study on Face Recognition Based on SVM of One-against-One and One-against-Rest Methods.” *Comparative Study on Face Recognition Based on SVM of One-against-One and One-against-Rest Methods - IEEE Conference Publication, IEEE,* 29 Jan. 2015, ieeexplore.ieee.org/document/7024355/.

[15] Manning, Christopher D., et al. *Introduction to Information Retrieval.* Cambridge University Press, 2018.

[16] Hanin, Boris, and Mark Sellke. “Approximating Continuous Functions by ReLU Nets of Minimal Width.” *ArXiv.org* 10 Mar. 2018, arxiv.org/abs/1710.11278.

5 Author Contributions

Tannavee Kumar: I was one of the project managers, I helped spearhead the agenda, facilitate meetings and workload, and make sure that everyone was on par. I also organized the writing of the paper, including doing the preliminary research having to do with types of machine learning models. I wrote the Abstract, Introduction, the Discussion, and parts of every section, proofreading, converting to Latex, formatting the references. I was also on the UIUX team, and helped come up with the general goal of the website as well as designing the website through photoshop to eventually be integrated for the frontend. I was also on the machine learning team and helped with the research as well as testing models.

Matthew Marlow: My role in this project was the UX/UI Team leader and a member of the ML/Algorithms team. In my role as the UX/UI Team leader, I oversaw, directed, and contributed to the creation of the front end (website and communication with the back end). As a member of the ML/Algorithms team, I wrote and executed the parameter sweep (ANN_paramsweep.py) for the creation of our FFNN and production of Figures 2.1 and 2.2. In addition, I worked with the ML/Algorithms team to come up with the theory behind our model.

Jiahui Dai: I was a member of Back-end and QA teams. I built the framework of back-end, setted up the flask server for back-end, took care of the communication of front-end and back-end, and documented the back-end code.

Jose Torres: I was in charge of the Data Management team. Our team was mainly responsible as acting as an intermediary between the data itself and the ML/Algorithms team. My role specifically was to talk to the ML team to determine what they needed and then coordinate among my team to get whatever the ML team needed. Alongside my team I implemented the data interface and also the DataBase that was later used in the project. Aside from that, I was also a member of the QA team. In this team I was responsible for documenting code for the Data Management team. Aside from that I wrote the DataOverview.md that goes into detail about what our data is. I also contributed to the data management aspect of the report and the misclassification bar plots.

Spencer Grossarth: I was the Backend team lead. As the team lead I coordinated the efforts of those on the Backend team by conducting the Backend meeting each week. Additionally, I helped determine the technologies that should be used to implement the server and what the general design of the backend should be. In terms of implementation, I was responsible for deploying the team's code onto a remote server. I created a custom Docker image that built and ran the frontend and the backend. After getting the Docker image working locally, I created an Elastic Beanstalk project and deployed the Docker container to it. I contributed to the report by writing the section on deployment under the Software Engineering heading.

Luc Nglankong: I was a member of the UX/UI and Software Engineering teams. On the UI team, I implemented the finalized designs of the website. I stylized both the input and results pages using React CSS and JSX code. I also added functionality for the "Feeling Lucky" button, "Random Search" button, the YouTube Search link and the loading screen. I also created the output text fields on the results page, including the predicted genre and probability, actual genre and probability, song rank and model rank. On the Software Engineering team, I pipelined the backend to connect the front end, data management, and ML team's hard work together. The pipeline receives input from the front end, queries the database, predicts the data using the neural network and send data to the front end in Flask. On the report, I documented sections 1.3(Polishing and incorporating design elements) and 2.4 (Pipeline). I also assisted the ML team by generating figure 3 (final model test/train accuracy vs epoch on small, medium and large datasets). When I joined the group late, I received plenty of extraordinary assistance to help me catch up from Nick, Tannavee, Jiahui, Cameron, Matthew, Spencer and Jose.

Jatin Mohanty: I was the leader of the QA team as well as a member of the Data Management team for this project. As leader and member of the QA team, I oversaw the writing of the documentation in the readme files as well as tested the code to make sure it functioned properly in the machine learning, data management, front end, and back end areas. As a member of the data management team, I created a database in SQLite with queries to take data from the database as well as contributing to help Jose and

Chance with the data interface for the ML team and Back end teams. In addition to this, we analyzed which data should be used and what transformations should be performed on them if any. Finally, I contributed to the Data Management section of the paper regarding the overview and preprocessing of the data.

Kevin Lee: I was the leader of the Machine Learning team and a member of the UI team. In the UI team, I helped design and create the front end along with the other members. In my role as the ML lead, I helped direct group members and ensure effective communication between members. I also talked with the Data Management team about the features and shape of the data, and Jose was instrumental in helping us understand the features. I helped test out and prototype other variations of ANNs that did not perform as well as the final. Further, I, along with the rest of the ML group, discussed and decided what approaches we wanted to try to use to optimize the model and what the final model should be. I also built the SVM and logistic regression models that we used as comparison to our final FFNN. In the paper, I contributed to Figure 1, the SVM, and logistic regression parts of the paper.

Chance Stewart: I was part of the Data Management and Machine Learning Algorithms teams, and I mostly worked on feature selection, running univariate selection with f-classif and mRMR selection using an executable program, as well as creating functions in the CSV interface that returned the column indices of feature selections for easier testing. I also worked on a wrapper for creating k-fold data splits (unused). I contributed to this paper by writing about the feature selection, as well as writing the analysis of figure 4. I repurposed the code used for training and testing models and changed it to create code that can take in multiple models and compare them, outputting the mean ranking with each successive iteration of training, which was originally used to produce figure 4. Later this was replaced with the current figure 4, which I also helped to write the code for.

Nick Stapleton: Project Management: I helped organize the original ten group members, coordinated weekly lead team meetings, delegated high level tasks to team leads, and maintained team online resources such as slack and github. I organized the project into 5 teams and assigned everyone to 2 team roles each. Then I assigned one person from each team as team leader that's responsible to attend a weekly meeting with the Project Managers and communicate on behalf of their team. Roles: Project Management, UI/UX Data Visualization (Front End), Software Engineering (Back End), Data Management, and Quality Analysis. The topic of music genre classification was my suggestion and I identified the free music archive data set on the UCI machine learning archive. Software Engineering: I shared my code from homework 2 and modified it to allow a FFNN to be parametrized, trained, then saved to disc for web deployment. I defined the interface that was used in the back end pipeline to return a result to the front end interface from the Data Management team's code to the Software Engineering team's code. ML Algorithms: I shared my manual one hot encoder from homework 2 used for neural network classification predictions. I contributed to the discussion on theory behind our model. I verified that the feature combination of mel-frequency cepstrum coefficients and spectral density produced the best results.