Music Genre Classification

Cindy Fang
Computer Science
Ryerson University
Toronto, ON
cindy.fang@ryerson.ca

Fatima Farishta
Computer Science
Ryerson University
Toronto, ON
ffarishta@ryerson.ca

Dylan Rodrigues
Computer Science
Ryerson University
Toronto, ON
d1rodrigues@ryerson.ca

***Abstract*— A final report that collates the results of three different machine learning algorithms to determine which of them can most accurately predict the genres of different music samples. This goal is intended to facilitate music classification, to enable the creation of better music-related applications that cater to the needs of consumers. To accomplish this goal, we selected three models we thought would be most suitable - neural networks, k-nearest neighbours, and logistic regression - and an appropriate dataset. After doing preprocessing on each of these models, we evaluated the initial accuracy scores on the train, test and validation sets and noticed some parameters we could use to improve performance. Finally, these steps allowed us to gain high accuracy scores for most of the models; we also outlined our future plans if our project was continued.**

I. Introduction

Music continuously evolves with its contemporary cultural atmosphere. Minor changes in properties such as rhythm, timbre, or pitch can drastically transform how we perceive a piece of music. With current technology, we can break down these high-level semantic labels into the low-level properties of their sounds. Machine learning models used in conjunction with these features can allow us to conduct deep analysis on these musical characteristics. A key classifier of music is the genre. Machine learning models can automate the task of detecting patterns of audio segments and classifying them into their respective categories.

Automatic music classification is cheaper, faster and reduces human error. Therefore, it is beneficial for companies to harness these properties to tailor their services to user preferences. We can sort, analyze music recordings, and store them into databases. Companies such as Spotify can create recommendation systems to predict music that a client may enjoy or create personalized playlists. As the list of applications continues to grow, so does the need to generate high-performance models.

Within this report, we will explore machine learning models and optimize them to classify music genres. Our analysis begins with choosing appropriate models that meet the requirements of our dataset and its features. Initially, we will create a baseline model and evaluate how well it works with the data. Afterwards, we will evaluate our outcomes and run a continuous cycle of training, evaluating, and fine-tuning each model.

A. *Challenges*

Music genre classification becomes challenging due to its subjective and sometimes ambiguous nature. Humans may find it hard to differentiate punk from rock music; likewise, classifiers face the same issue. A particular song may be a combination of genres or exhibit characteristics that overlap with other genres. Music is sensitive to change, the sounds or formal definitions of any genre can vary in different eras and cultures.

B. *Prior Work*

Due to the demand, there is extensive research done on the properties of music genres. A notable study done by Ezzaidi and Rouat uses MFCC from the average spectral energy of split music pieces to assist in a 99% accurate recognition system [4]. Another research using the same GTZAN dataset done by Ahmed, Paul, and Gavrilova applies their methods based on texture analysis and visual representation of audio signals, along with the use of gradient directional patterns and computed spatial histograms, to send to support vector machines to classify [3]. Furthermore, research with a Latin dataset (3160 songs with 10 genres) was done by Silla, Koerich, and Kaestner using binary classifiers. In this case, the results went through space and time decomposition to obtain a further set of individual results. These were sent to machine learning algorithms such as naive Bayes, decision trees, k-nearest neighbours, and support vector machines [6].

II. Problem Statement

The goal is to use classical and deep learning models to classify the GTZAN dataset song features into 1 of the 10 music genres. For example, given a list of 59 features from an unclassified song, applying machine learning models is essential to accurately sort the song into its proper genre. This process consists of preprocessing the dataset and then splitting the already feature-selected dataset into training, testing, and validation datasets. Next, machine learning techniques will be applied to evaluate and train the applied models to classify each features list with high accuracy. Extra evaluation techniques such as error, accuracy, precision, f1, recall scores and confusion matrices, along with various hyperparameter finetuning techniques will be used on the data to analyze and upgrade the performance of the models in detail.

Code: https://github.com/ffarishta/Music-Genre-Classification

A. *Dataset*

We will be using the GTZAN music classification dataset from Kaggle [1]. The GTZAN dataset is the most used public dataset for music genre recognition research. It consists of a collection of 10 genres with 100 audio 30-second files each (with a total of 1000 audio files). The 10 genres are Blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. All tracks are 22050 Hz Mono 16-bit audio files in .wav format. Additionally, it contains Mel Spectrograms and 2 CSV files containing features of the audio files are also present in our dataset. The 3 second variation of the CSV file contains features with the dimensions of (9990, 60) and the file with the 30-second data has dimensions of (1000,60). The audio files are split into 3-second intervals to produce more data for training.

III. Methods and Models

We applied similar preprocessing methods to our dataset to create a fair baseline for our models. Then we individually constructed variations of each machine learning model. According to our analysis metrics, we needed to finetune the models until we were satisfied with the outcomes. Our models include: (i) Linear Regression, (ii) K-nearest neighbours, and (iii) Neural Networks.

A. *Pre-processing and Features*

Since this GTZAN dataset contains all the features required for the classification, we will not be extracting any features directly from audio files. The dataset has 60 features from the audio signals present in the recordings of the dataset. These include both time-domain and frequency-domain audio features. Examples of some features include zero-crossing rate, spectral centroid, MFCCs, chroma vector, harmony, tempo, etc.

Since most models do not work well with semantic training labels (e.g., “blues”), we will transform them using one-hot encoding. This method can help remove implicit assumptions about each class in relation to other classes, such as two nearby values being more similar compared to ones further apart [2]. The features in this dataset vary greatly in magnitudes units and scales; therefore, scaling will allow the models to make better predictions. We apply a min-max scalar to transform all numerical data into values between the range of 0 to 1. Additionally, the data is split into a ratio of 70:20:10 % for training, testing, and validation sets, respectively. We use the validation set to tune the hyperparameters during our analysis and the test set to evaluate the final performance of the model. Along with the preprocessing steps we already mentioned, we also dropped any columns that came with the dataset but were not necessary for the model, such as the “filename” and “length” columns.

B. *Neural Network*

Neural networks perform well with a variety of classification problems. For the baseline model, we created a multi-layer fully connected neural network and experimented with different properties. Our final baseline model consists of 5 layers, these layers have 512,256,128,64, 10 neurons respectively. We decided the first four hidden layers will use ReLU, and the output layer will use SoftMax as the activation function. We chose ReLU because it reduces sparsity and the likelihood of vanishing gradients. Since we are doing multiclass classification, our loss function is categorical cross-entropy with an Adam optimizer. The initial model has a high variance problem, we will discuss in later sections how we plan to use L1, L2 and dropout layers to fix this issue. The following figure is the neural network architecture for our baseline model.

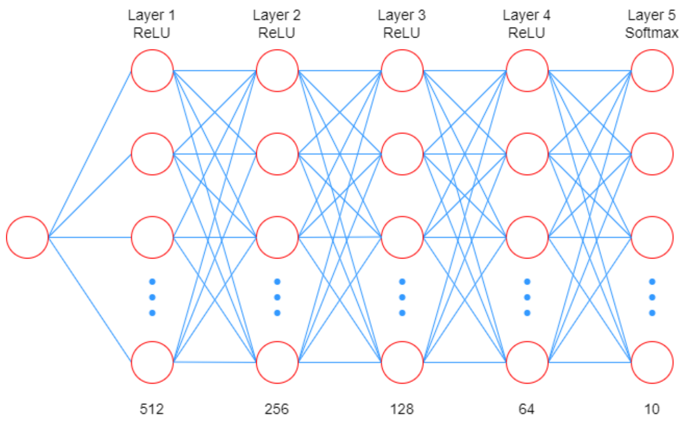

Fig. 1. Neural network architecture for baseline model

C. *Logistic Regression*

In Logistic Regression, we use the sigmoid function (as seen in Eq. 2) as a hypothesis function. Then by using gradient descent to decrease the log likelihood to fit the parameters

$$S(z) = \frac{1}{1+e^{-z}} \tag{2}$$

We considered using the logistic regression algorithm because it is a classifier that works well when an algorithm is required to choose the outcome of a variable. We thought it would be well suited in this scenario; we required a model that would decide what genre a piece of music belongs to based on the features, or decide what value the genre variable would be and a classifier such as this model would suit these needs

Next, we created a multinomial logistic regression model rather than a binary logistic model because the target variable, being the genre of a piece of music, had more than one value. For the baseline model, we initially set the max iterations to 10000 so the model can reach convergence. Moreover, we used the lbfgs solver in both cases, which implemented the LBFGS algorithm for optimization. To increase the accuracy, we then fine-tuned hyperparameters such as the sample weight when calling the function for fitting, and the amount of regularization, the C value, during the testing step.

D. *K-Nearest Neighbour*

The K-nearest neighbours model assumes that all similar data points are close to each other in distance. This ‘lazy’ model does barely any training to the dataset to make generalizations of the genre classification. To implement this model, we first initialized the K value and calculated the distance between each query and current example from the dataset. After storing the results from shortest to farthest distance, the first K entries would be selected to compute the mode of the K labels. For our baseline model, we selected 11 as the initialized K value. After applying performance metrics and grid search to determine the optimal K value for the highest accuracy, 2 as the K value, along with the weights being distance and metric being "Manhattan" (as seen in Eq. 2) were applied for the final model. We chose this model because of its compatibility with multiclass classification, its low calculation time, ease to interpret its results, and its predictive strength.

$$d(x,y) = \sum_{i=1}^{m} |x_i - y_i| \quad (2)$$

E. *Performance Metrics*

The metrics we used to implement functions that assess model prediction error and success, along with creating visualizations to get a better understanding of model performance through training and finetuning. The following performance metrics are used to evaluate and compare model performance:

1. Accuracy score
2. Precision
3. Recall
4. F1 score

To determine the overfitting or underfitting within the models, accuracy and error scores on the training, testing, and validating data sets are very important. The error scores are essential in showing bias and variance errors and making decisions when dealing with bias-variance tradeoffs. The confusion matrix is used to predict the accuracy of the model identification of the data and its correct genre. Graphs are plotted to provide a visualization of accuracy and error trends for each model. By finding the precision score, we can see the quantitative existence of true positive and false positive predictions; for recall score, we can ascertain true positive and false negative predictions. Next, with the f1 scores, it would take both precision and recall scores into the equation and provide a mean where model performance gets better as the score is closer to 1, and worse when the score is near 0.

IV. Results and Descussion

We evaluated each model on the train, validation, and test sets with all the performance metrics mentioned in previous sections.

To improve the performance of the models we use hyperparameter finetuning, notably regularization. The following figures showcase the improvements on accuracy from the baseline to the final models.

Model	**Train Accuracy (%)**	**Dev Accuracy (%)**	**Test Accuracy (%)**
KNN	82	76	76
LR	69	67	69
NN	94	84	86

Fig. 2. Accuracy for baseline models

Model	**Train Accuracy (%)**	**Dev Accuracy (%)**	**Test Accuracy (%)**
KNN	99	91	93
LR	75	72	73
NN	92	89	90

Fig. 3. Accuracy for finetuned Models

A. *Neural Networks*

The accuracy of our baseline on the training set is roughly 95%, but 82% on the validation set. Fig. 1 illustrates the divergence of both accuracy and error of the validation set from the training set after roughly 20 epochs. The separation is an indication that the model is suffering from overfitting resulting from high variance. Although high variance may be indicative of noise in the data set or the lack of training data overall, we can still balance it by reducing the complexity of our model. Using both L1 and L2 regularization (at rates of 0.1, 0.01, 0.001, 0.0001), we found that it did reduce the variance but at a significant cost of the accuracy. Since the reduction in accuracy was too significant, so we decided to try adding "dropout" regularization layers. Dropout regularization can reduce the complexity of the model by shutting down neurons during particular training steps [5]. After setting the dropout rate to 0.3, it performs better than L1 and L2 as it didn't sacrifice accuracy while also reducing our variance problem.

A problem with many neural networks is that the gradients of the layers either become too small (vanishing) or too large (exploding) after certain iterations. Small gradients leave lower layers connection weight virtually unchanged allowing the model to never fully converge. On the other hand, large gradients can make the model begins to diverge [5]. A method to solve this problem is to consider how to initialize the weights. In our case, we use the "he-initialization" method and change the activation function of the hidden layer to the ELU function. The accuracy on the test set improves from 84% to 90%.

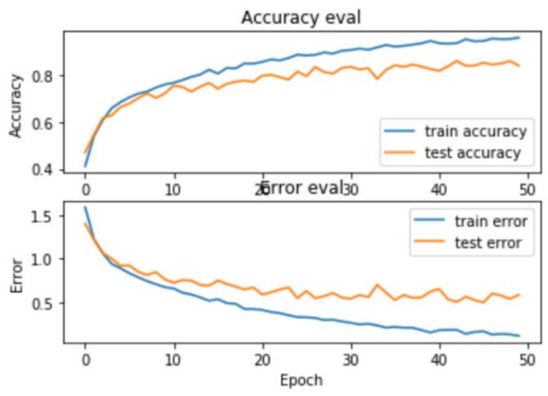

Fig. 4. Accuracy and error for baseline model after 50 epochs

Code: https://github.com/ffarishta/Music-Genre-Classification

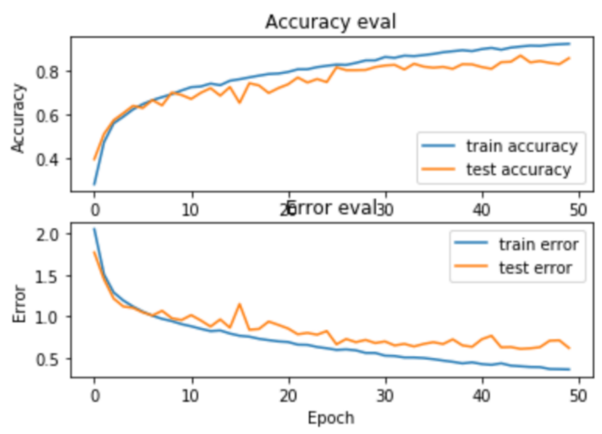

Fig. 4. Accuracy and error using L1 and L2 regularization after 50 epochs

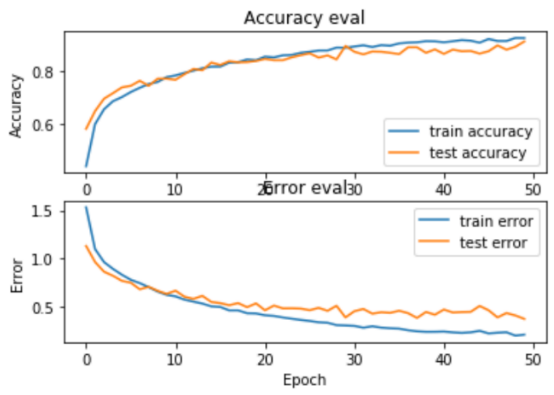

Figure. 5. Accuracy and error using dropout regularization and He-initialization after 50 epochs

B. *Logistic Regression*

The accuracy of the baseline model on the the training and testing were very close in terms of accuracy but performing poorly.

One issue with logistic regression is that it is often susceptible to high bias and low variance when dealing with complicated datasets. Similarly, our baseline model suffers from underfitting as it has a 7.24 error rate on the test set. Our dataset contains complex features, which may be a disadvantage of this model. To solve this issue, we could incorporate more hyperparameters to increase the overall complexity of the model. After conducting grid search cross validation on the baseline model, we found two hyperparameters we could use to do some fine tuning; the C value, which controlled L2 regularization, and the number of iterations.

Accordingly, we manipulated the C value of the logistic regression model during the second and final version. The C value determines the amount of regularization that occurs, inverse to the amount entered. If we increased the regularization strength to 9.0, it should in turn decrease the bias. Additionally, when it came to choosing 1000 and 10000 as the number of iterations, either had a negligible impact on the accuracy, so we simply went with 10000 to forestall any issues when compiling our code.

Finally, we noticed that increasing the sample weight when fitting the data on the training set gradually increased the accuracy of the model. Therefore, we increased the sample weight to 20,000. Along with the regularization step above, the change in sample weight allowed the second model of logistic regression to show a slight increase in efficiency, from around 69.36% to approximately 73.06%. We have included the confusion matrix and classification report for the second, streamlined model below.

Hyperparameter	Baseline Model	Final Model
C	1.0	11.0
max_iter	1000	10000
class_weight	None	Balanced

Fig. 6 Hyperparameters for both baseline and final models

C. *K-nearest Neighbours*

As seen below, the accuracy and error scores for the baseline model were low. Specifically, the testing dataset has an error rate of 3% while the validation dataset has 6%. The difference between the scores shows that the model is overfitting and high in variance (and low in bias). This means that there will likely be significant changes to the model's predictions with changes to the training dataset.

To reduce the high variance, we decided to change the K value to its optimal value, as a way of fine tuning the hyperparameters. The grid search function was used to determine the best metric, K value (number of neighbors), and weights that resulted in the highest accuracy of the model. The results of the grid search are shown below, along with a much higher accuracy score as well improving from 76% to 93%.

Hyperparameter	Baseline Model	Final Model
K	11	2
Metric	Euclidean	Manhattan
Weights	Uniform	Distance

Fig. 7 Hyperparameters for both baseline and final models

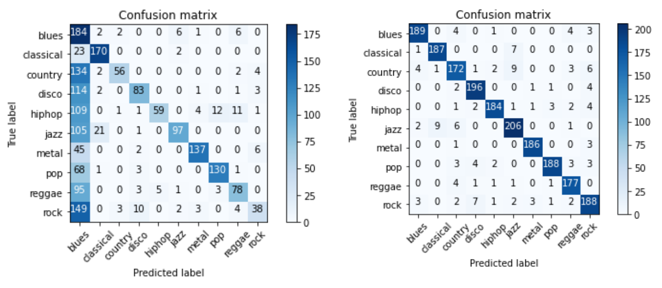

Fig. 8 Baseline and final model confusion matrices for KNN

Code: https://github.com/ffarishta/Music-Genre-Classification

D. *Future Changes*

Going forward, we would appreciate the opportunity to integrate our work into an actual application. There are many uses for models that deal with music genre classification. We would like to create an application that can recommend music to users based on the genres they already listen to frequently. For this goal, a model that can efficiently categorize a piece of music based on its characteristics is extremely useful. Moreover, if our project were to be continued, we would try out more complex models such as CNN (convolutional neural networks) and SVM (support vector machine). We would also do some of our own in-depth exploratory analysis of different music samples instead of just relying on datasets created by others. This would help us do our own feature extraction and decide what key features of the data to operate on. It would also allow us to include many genres, such as the currently very popular Korean pop genre. Finally, we would consider doing feature refining to maximize the efficiency and accuracy of the models we use.

V. Implentation and Code

We initially did all of our experiments on Jupiter notebook to conduct analysis and then transferred them to python scripts. The traditional classifiers are created using sklearn libraries, and we used Keras for the neural network. Additionally, we used matplotlib to construct our diagrams.

We used references from many papers and blogs to generate ideas for models and techniques.

VI. References

[1] Ahmed, Faisal & Paul, Padma Polash & Gavrilova, Marina. (2016). Music Genre Classification Using a Gradient-Based Local Texture Descriptor. 10.1007/978-3-319-39627-9_40.

[2] A. Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. (First ed.) 2017.

[3] A. J. H. Goulart, R. C. Guido and C. D. Maciel, "Exploring different approaches for music genre classification," Egyptian Informatics Journal, vol. 13, (2), pp. 59-63, 2012.

[4] Ezzaidi H, Rouat J. Automatic musical genre classification using divergence and average information measures. Research report of the world academy of science, engineering and technology; 2006.

[5] G. E. Hinton et al, "Improving neural networks by preventing co-adaptation of feature detectors," 2012.

[6] Silla, Carlos & Koerich, Alessandro & Kaestner, Celso. (2008). A Machine Learning Approach to Automatic Music Genre Classification. J. Braz. Comp. Soc.. 14. 7-18. 10.100

Code: https://github.com/ffarishta/Music-Genre-Classification