Music Genre Classification
Model Performance Report

Edwin Low

1 Data Exploration

The objective of the project was to classify 30 second clips of music in a test set based on a set of training data of the same format. The files were ".wav" files, associated to one of the genres ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']. After listening to some files and trial and error during feature extraction, we realized that the average features across a 30 second .wav file do not accurately describe the genre of music associated with that file, as there can be a lot of variation over a long time. Thus, we thought of splitting each .wav file into short clips of equal length, then predicting the genre for each 30-second audio file by 'taking a vote' from the predicted labels of the shorter clips. In other words, we took the mode of the predicted labels for each 30 second file.

To test the accuracy of our models, we generally split our training data into 80% train and 20% validation data, before submitting our predictions from fitting the model to all training data.

2 Feature Extraction

We made use of the librosa library to extract features from each audio file. After trying several features, we decided to extract a combination of Chroma, MFCC, and Spectral features. Since these features are vary with time, we took the time-average of the features over a small period of time.

In addition, we normalized the train and test datasets of extracted features by using the StandardScaler() function from Scikit-learn.

When it comes to machine learning, more data is usually better. So, though not a feature in and of itself, one method we leveraged when extracting the features was splitting the audio into smaller clips. In particular, we split each of the provided 30-second ".wav" files into ten 3-second clips. The choice of 3 seconds was somewhat arbitrary; however, we hoped that this number would succeed at providing enough data while not being so short that our models would be unable to discern their genre. As a result, our models had ten times the amount of data to work with – hopefully granting them a better understanding of the patterns for each genre. In our final submissions we further shortened the clips to 1.5 seconds to get even more localized data in a large quantity. We will compare the performance of the different feature extractions later in the SVM section.

2.1 MFCCs

Mel Frequency Cepstral Coefficients are coefficients calculated by the following steps:

1. Compute the Fourier transform over a short time window of the signal.
2. Map the power spectrum onto the Mel scale from the standard frequency, with relation given by $m = 2595 \log_{10}(1 + \frac{f}{700})$.
3. Take the logs of the power at each Mel frequency.
4. Take the discrete cosine transform of the calculated Mel power spectrum.
5. The resulting coefficients are the Mel Frequency Cepstral Coefficients.

The benefit of employing the Mel frequency scale is that this scale more accurately models how the human auditory system distinguishes frequency.

Figure 1 shows the spectrum calculated from the MFCC extraction for a 30 second audio clip.

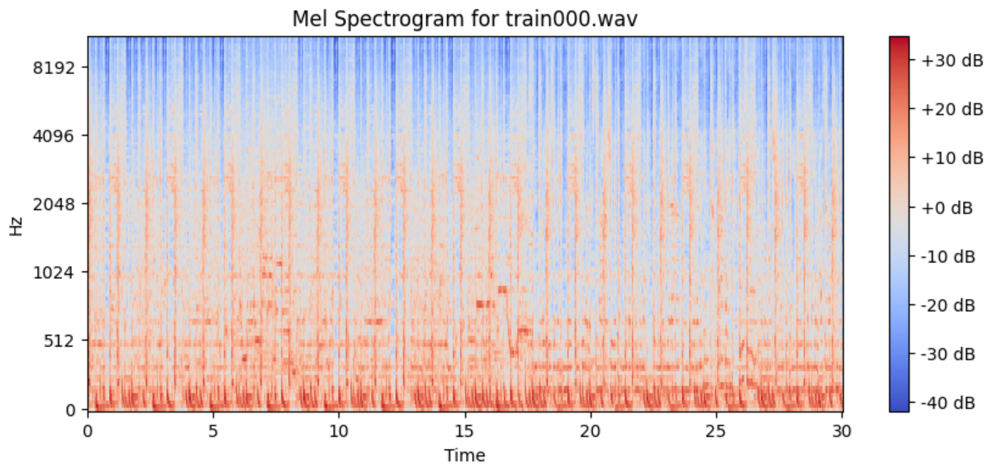


Figure 1: Mel spectrogram for train000.wav, using librosa.

2.2 Chroma

Chroma features are another form of spectral information, but do not distinguish between octaves, that is, the information is solely focused on the pitch in a short time frame (in the scale of ms). This means that the spectral information is distributed to 12 bins (columns in our data matrices) associated to the 12 tones (A, A#, B, etc.) in Western music. Figure 2, the Chroma spectrogram is shown for a sample audio file.

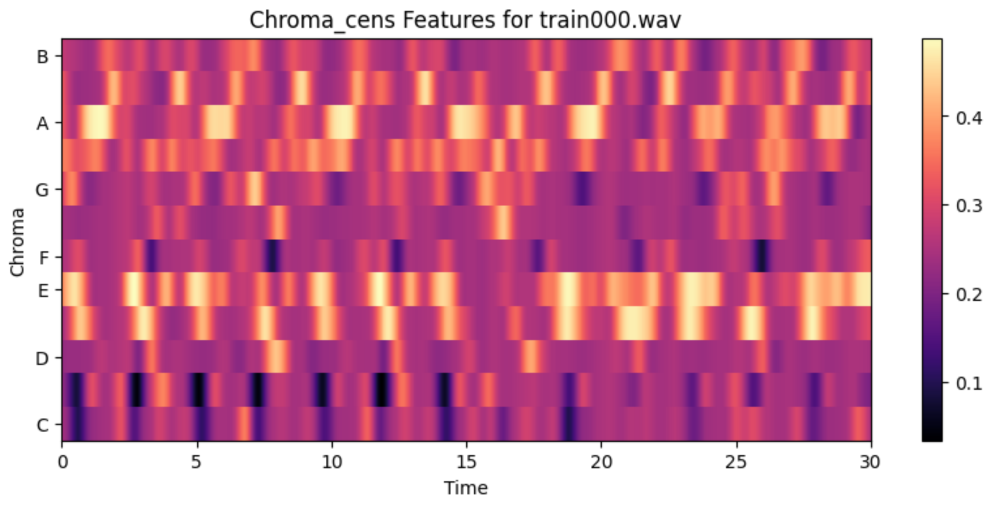


Figure 2: Chroma spectrogram for train000.wav.

2.3 Spectral Contrast

For spectral contrast features, the spectogram is divided into sub-bands (see Figure 3) and the spectral contrast is computed by comparing the mean energy in the top quantile frequency and the mean energy in the bottom quantile. Higher contrast corresponds to narrower band signals.

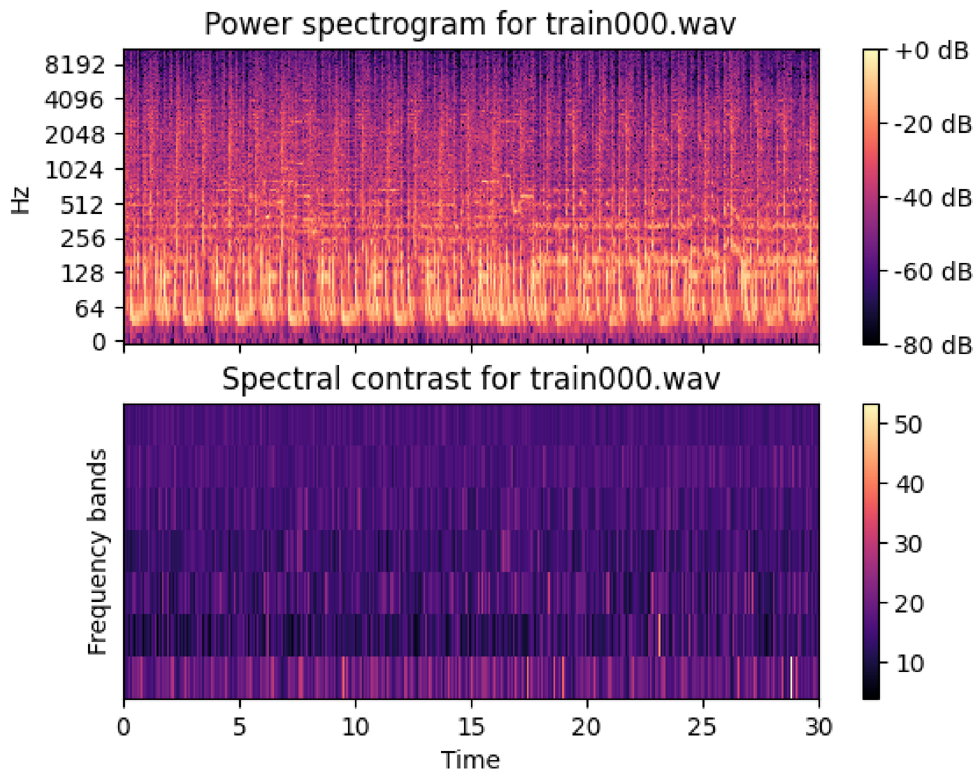


Figure 3: Spectogram divided into sub-bands.

2.4 Spectral Flatness

Spectral flatness, also known as the tonality coefficient, describes how noise-like a sound is, as opposed to pure tones. Its value ranges from 0 to 1.0, with 1.0 representing the value for white-noise (energy is constant with respect to frequency).

2.5 Additional Remarks

We considered the use of other features such as tonnetz and tempo features, however we discarded each for the following reasons: tonnetz features added too many columns to our data matrix and we felt that we had enough data already for spectral information, and we judged that tempo features would not be as useful when tempo varies a lot even for given music genres. In terms of the shortening the audio clips, we started by splitting each .wav file into ten 3-second clips, and then later on found out that our models produced better results with shorter 1.5 second clips, which in turn doubled the amount of data to work with.

3 Model Selection

3.1 KNN (K-Nearest Neighbors)

As a simple working model, we started off by using a KNN algorithm to predict the test labels. KNN is an algorithm that takes the mode value of the k nearest neighbors of a data point to investigate as the associated data label. We used Scikit-learn's KNeighborsClassifier() function to employ the model, and tested several values for k. Figure 4 shows that we get the highest accuracy on the validation set for the lower k values. However, this will overfit the model to the training data because it is overly dependent on individual data points from the training set rather than observing any trends. On the other hand, choosing k too large will

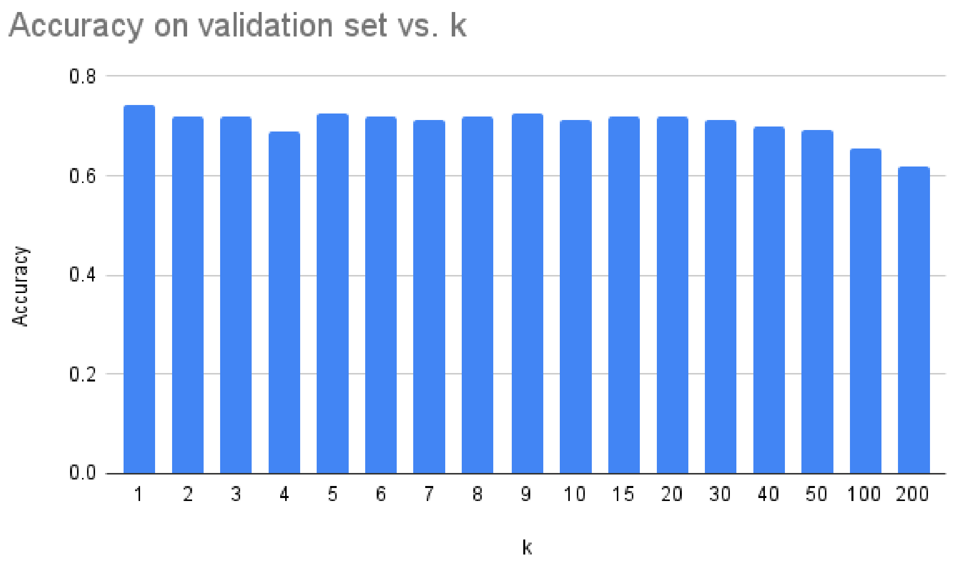


Figure 4: Accuracy of KNN on validation set (20% of train set) for varying k values.

sacrifice the accuracy because taking farther points into consideration will inevitably lead to taking the data points with the wrong labels into consideration.

3.2 SVM (Support Vector Machine)

3.2.1 Algorithm

Kernel Support Vector Machines (Kernel SVMs) can distinctly classify data with complex boundaries. Using a kernel, SVMs can project data into a higher-dimensional space to improve separability. The algorithm defines a hyperplane with $f(x) = wx+b$ to maximize the margin ($\frac{1}{\|w\|}$) and enhance stability. The optimization problem, considering misclassifications with slack variables ξ, is given by:

$$\min_{w,b} \left[\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \right]$$

where C moderates the margin and misclassification trade-off.

3.2.2 Design Specifications

1. Kernel choice in SVMs affects the feature mapping and classification accuracy:

   - Linear Kernel: The simplest kernel, suitable for linearly separable data, is defined by the dot product $K(u,v) = u \cdot v$. It does not involve any mapping to a higher-dimensional space and is computationally efficient.
   - Polynomial Kernel: It adds flexibility by allowing the decision boundary to curve, capturing more complex dataset structures. The polynomial kernel is defined by $K(u,v) = (u \cdot v + r)^d$, where d is the polynomial degree and r adjusts the influence of higher-order terms.
   - Radial Basis Function (RBF) Kernel: Also known as the Gaussian kernel, the RBF kernel can handle an infinite-dimensional space and is defined by $K(u,v) = e^{-\gamma\|u-v\|^2}$. The γ parameter determines the distribution's spread, influencing the smoothness of the decision boundary.

- Sigmoid Kernel: Similar to the neural network's sigmoid function, it is given by $K(u, v) = \tanh(\gamma u \cdot v + r)$. This kernel's parameters γ and r control the function's shape and slope, though it is not commonly used due to potential convergence issues.

Each kernel type balances data fitting and generalization, impacting SVM performance with regards to overfitting and model complexity.

2. The parameter C is the regularization parameter for the SVM. The strength of the regularization is inversely proportional to C. Decreasing the value of C increases the regularization strength, which will create a smoother decision boundary and potentially underfit the model. Increasing C decreases the regularization strength, allowing the model to fit more closely to the training data, which may lead to overfitting.

3. The `decision_function_shape` parameter determines the shape of the decision function. The option 'ovo' stands for one-vs-one and results in a decision function of shape $\frac{n(n-1)}{2}$, where n is the number of classes. This approach fits a separate classifier for each pair of classes. It is often preferred for multi-class classification problems.

4. The parameter γ in the RBF kernel determines the influence of individual training examples. A lower γ value means a longer reach for the training example and a smoother decision boundary. A higher γ value means a shorter reach and a decision boundary that more closely fits around the data points. When set to 'scale', γ is computed as $\frac{1}{n\_features \times \mathrm{Var}(X)}$, where X is the feature matrix.

3.2.3 Feature Selection for SVM

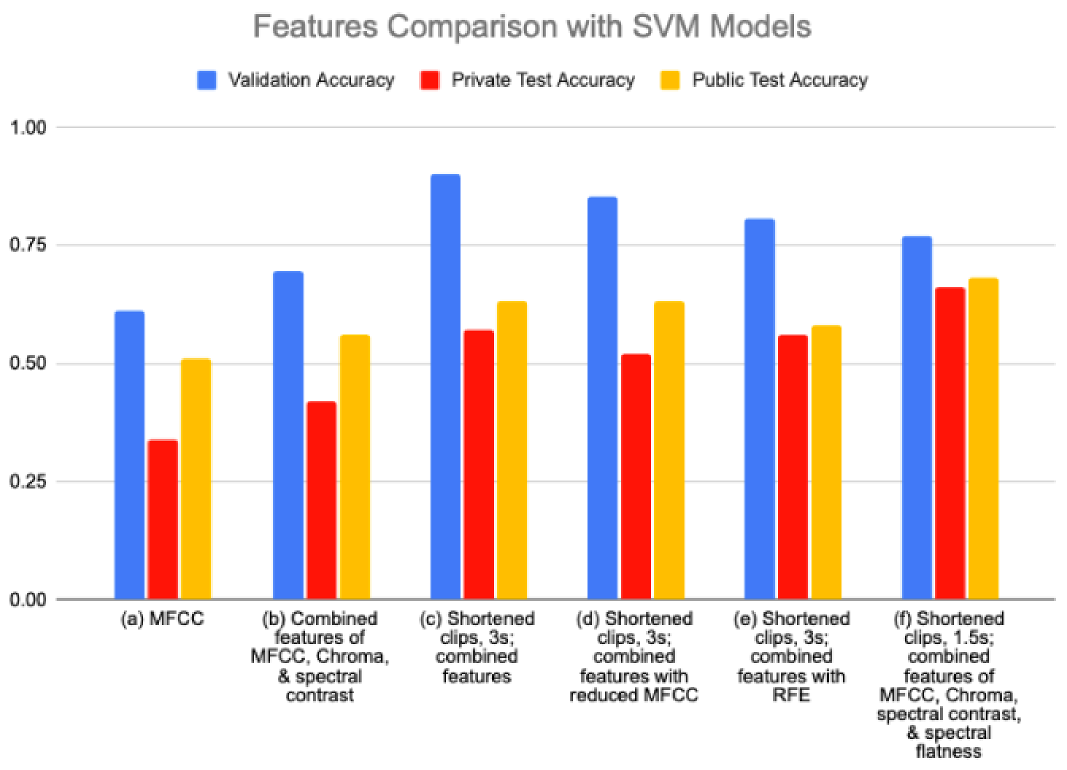


Figure 5: Features Comparison with SVM Models. The optimal hyperparameters for all these models are determined with GridSearchCV.

In Figure 5, the SVM model is tested with different extracted features. From left to right, model (a) uses 40 MFCC features extracted from the training audio files. Model (b) combines MFCC, Chroma, and spectral contrast features. Model (c) involves shortening the 30-second clips into 10 3-second clips, then extracting the MFCC, Chroma, and spectral features from these shortened clips. Model (d) involves the same procedures described in model (c), but the number of MFCC features are reduced from 40 to 25. Model (e) also involves the same procedures described in model (c), but now RFE (Recursive Feature Elimination) is applied until only 20 total features remain. Model (f) is the optimal SVM model used in our submission. It splits the audio files into 1.5-second clips and uses MFCC, Chroma, spectral contrast, and spectral flatness features. None of the features was reduced or eliminated.

- For model (a), with only 40 MFCC features, the training validation accuracy is quite low at 0.6125. There is clearly not enough information from the MFCC features alone. The private test accuracy of 0.34 (not visible to us before) also suggests that MFCC alone cannot classify unseen data well.
- Model (b) combines MFCC, Chroma, and spectral contrast to extract other features from the audio which may be relevant to classification. The accuracy increases a little, but there isn't enough data to work with from the initial 800 train files.
- Model (c) addresses this issue by splitting the audio into shorter clips while using the same combined features as before. This increase in the quantity of data boosts the training validation accuracy up to 0.900625. However, the public test accuracy is comparatively quite low (at only 0.63). This means that the model overfits on the training data and cannot generalize well to the testing data.
- To address issues with potential overfitting, we observe the features extracted and notice that there are too many MFCC features (40 of them) compared to other types of features. It is possible that the MFCC features "overpowers" the other features: the model may not account for other types of features enough. For model (d), the combined features only contain 25 MFCC features (instead of 40 in model (c)). However, this change decreases the train accuracy while keeping the public test accuracy the same. We decide that these additional MFCC features are actually necessary to keep the validation accuracy higher. Removing some of the MFCC features only decreases the validation accuracy and does not affect the public test accuracy. (The private test accuracy also proves our point: model (d)'s private test accuracy is lower, which means that all 40 MFCC features are important. MFCC features are not the culprit of overfitting.)
- Model (e) uses RFE (Recursive Feature Elimination) on the combined features from model (c). This is another effort to reduce overfitting. RFE is a feature selection method that fits a model with the existing features and removes the weakest feature. This fitting process is repeated until a specified number of features remains. In this case, the model is initially fit with all the combined features from model (c). RFE ranks the features by their importance to the predictive accuracy. The least important feature is removed recursively. The model is then refitted with the new, reduced set of features. This process is iterated until the specified number of features is reached. In this case, cross-validation is used to identify the optimal number of features, which is 20.

  Model (e) runs RFE and reduces it down to 20 features in total. The validation accuracy and public test accuracy are decreased slightly compared to models (c) and (d). The private test accuracy is comparable to that of model (c), although model (e)'s public test accuracy is lower. RFE slightly reduces overfitting on the private test data, but it worsens the model's prediction on the public test data. Overall, model (c), which does not employ RFE, still performs better.
- Based on the above, we determine that techniques to reduce overfitting in models (d) and (e) are not effective. They may slightly reduce overfitting but also decrease the public (and private) test accuracy. Therefore, we leave out these methods in our optimal model (model (f), in svm.ipynb).

  Model (f) uses 1.5-second clips instead of 3-second clips. MFCC, Chroma, spectral contrast, and spectral flatness features are extracted from these 1.5-second clips, and all the features are kept for the training. This model produces a lower validation accuracy compared to before, but the public/private test accuracy is higher, indicating that the model does not overfit on the training data. We successfully reduce the effects of overfitting by shortening the audio clips and giving the model more data to work with.

3.2.4 Hyperparameter Tuning

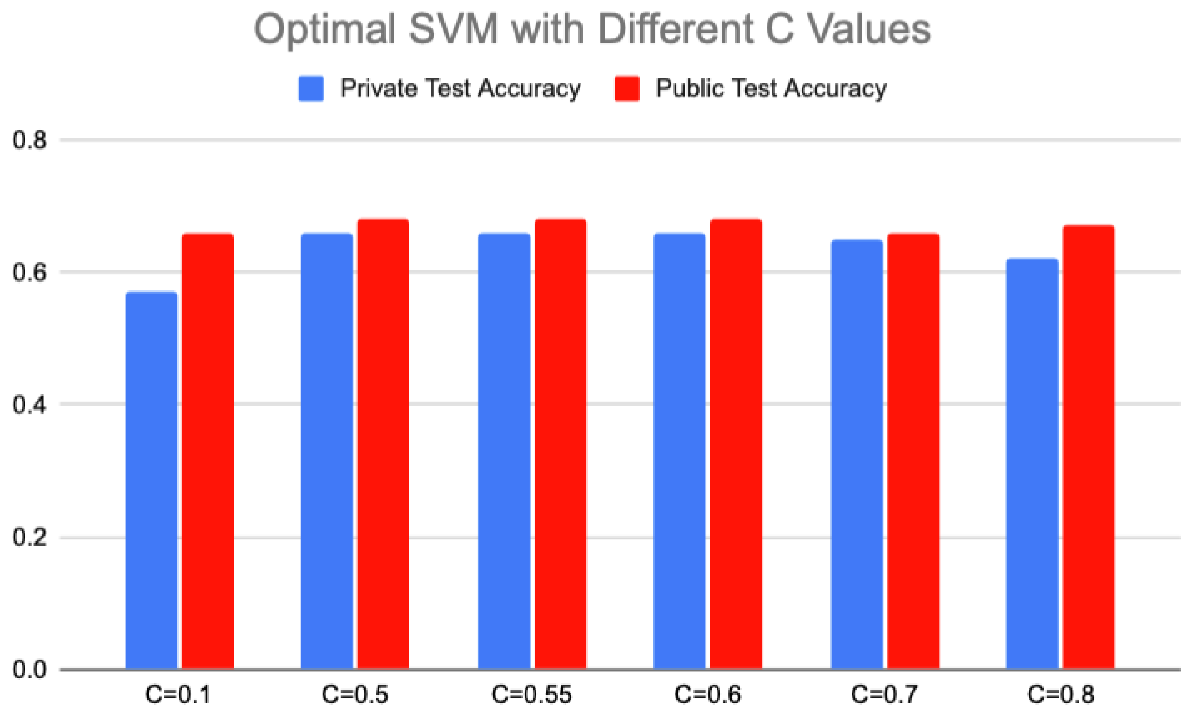


Figure 6: Optimal SVM with different C values

For the optimal SVM model (f), GridSearchCV is used to perform an exhaustive search over a specified parameter grid. It tries out every combination of parameters provided and finds the best combination. The parameters provided are C, kernel, and gamma, as detailed above in section 3.2.2. GridSearchCV returns that the best parameter combination is {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}.

In Figure 6 above, different C values are tested out on the model with gamma as 'scale' and an rbf kernel. A C value of 10 (the optimal value according to GridSearchCV) is likely going to overfit the data. It is desired to tune the C value to get the optimal test accuracy while not overfitting on the training data.

At this point, we have many submissions left on Kaggle, so we decide to try out many C values at around the same range. We randomly guess C=0.8, which gives us the highest public test accuracy on Kaggle so far at 0.67. We tune the C-value slightly and discover that they all give us similar public test accuracies. C=0.1 is too low because it decreases the public test accuracy. We determine that C=0.5 is the lowest C-value that can minimize overfitting and not sacrifice the model's public test accuracy.

Based on the private test accuracy, C=0.8 is too large because it still overfits slightly on the public test data. C=0.1 does not have enough samples as support vectors. The model does not classify enough training examples correctly, so it cannot generalize well to the unseen private test data. Its private test accuracy decreases. C=0.5, C=0.55, and C=0.6 gives the same public and private test accuracies. Our final selected model with C=0.5 gives us the best possible public and private test accuracies.

3.3 Neural Network

CODE HERE: `https://colab.research.google.com/drive/1LJrYwIWbrCV-hds3b1J23440IApDF4My?usp=sharing`

Another model that we implemented was a multilayer perceptron (MLP). MLPs are artificial neural networks consisting of layers of fully connected neurons. In general, MLPs do well at analyzing vectorized data, so although it is a relatively simple kind of neural network, we hoped that it would do well at classifying our data.

3.3.1 Model Architecture

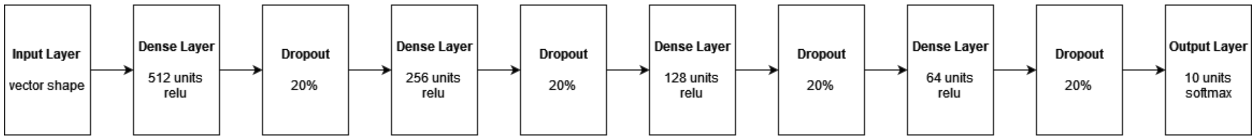


Figure 7: Model architecture for our multilayer perceptron.

As with any good multilayer perceptron, the meat of our model is in our hidden dense layers. There are four such layers, with the first having 512 neurons and this quantity decreasing by a factor of 2 until the last layer, which has 64 neurons. We found that these bounds worked best with this architecture, compared with either removing the 512-neuron layer or adding a 1024-neuron layer before it, and with either removing the 64-neuron layer or adding a 32-neuron layer after it. Each of these dense layers uses the activation function ReLU, which allows the model to find a nonlinear relationship between the data's features. The choice of ReLU is generally standard for these types of neural networks.

Between the dense layers are dropout layers. In a dropout layer, neurons from the previous layer are ignored ("dropped out") at some probability. This means that on each iteration, the model examines a slightly different set of weights. The choice to leverage dropout was to mitigate overfitting, as we noticed signs of overfitting when training our model without dropout layers.

3.3.2 Implementation-Specific Choices

- **Dense Neurons**: The number of neurons in our dense layers began at 512 and decreased by a power of 2 until reaching the dense layer with 64 neurons. The choice of these numbers was initially arbitrary (256 to 32), but we found that these boundaries worked best with our implementation compared to: adding a 1024-neuron layer at the beginning, or removing the 512-neuron layer; or adding a 32-neuron layer at the end, or removing the 64-neuron layer at the end.

- **Dropout Probability**: Dropout probability is a tunable parameter. A machine learning engineer will want a high enough probability that dropout will have an effect on overfitting, but a low enough probability that the model can still understand the relationship between the features of the data. In our case, we found that $p = 0.2$ was the sweet spot for all the layers, as higher and lower values showed worse accuracy.

- **Number of Epochs**: After training a few different versions of the multilayer perceptron, we found that the accuracy tends to converge around 80 epochs. We chose to go to 100 to give the model a little more leeway with when to converge. With this specific MLP architecture, however, the accuracy seemed to have mostly converged by 40 epochs; if we had more time, we might have considered experimenting with the number of epochs more to see if it had any impact on overfitting.

3.3.3 Compilation and Training

When compiling the model, we used the ADAM optimizer because it is known to achieve high accuracy quickly with multilayer perceptrons. Specifically, it can adapt the step sizes in training according to how quickly the loss is descending. This feature allows the model to bypass local minima in the loss function to (hopefully) find a better minimum.

We used sparse categorical cross-entropy as our loss function, because it is designed for multi-class classification problems. The only necessary adaptation was to encode the labels as integers (0-9).

During training, we split the input dataset into training data and validation data at a ratio of 8:2. This struck a balance between having enough training data for the model to actually learn with and having enough validation data for the validation step to be meaningful. On every epoch, the model first trained on the training set before validating its results on the validation set. This is useful for minimizing overfitting, since the model cannot train with the validation set but is still directly evaluated on its performance with it.

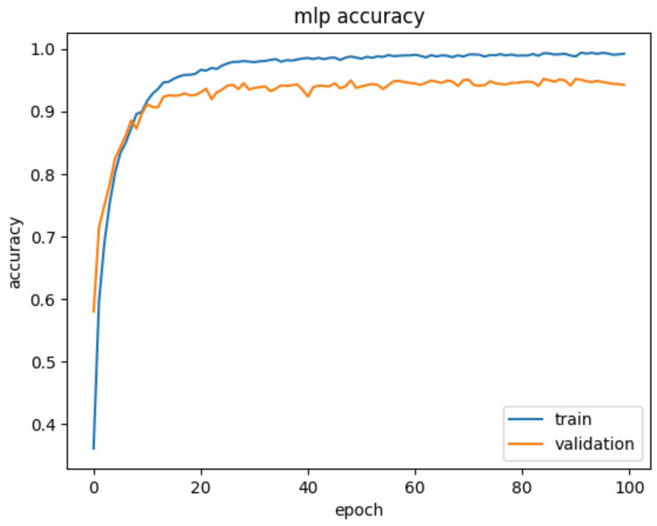


Figure 8: Training and validation accuracy of the MLP over time

3.4 Random Forest

The Random Forest classification model is closely related to the simple decision tree model. However, simple decision trees have the tendency to overfit the model to the training data. To circumvent this issue, multiple decision trees are used on a random subset of data over a random subset of features, hence the name “Random Forest”. The classifier then takes the mode predicted label from these decision trees to come to a final decision. By taking the results from multiple decision trees, the model works around overfitting since each tree is trained on a random subset of the training data. Thus, the accuracy is generally higher than traditional decision tree methods.

3.4.1 Algorithm

Below is an explanation of some critical methods used in random forest classification.

Bagging

Bagging, aka bootstrap aggregating, is a crucial technique used in random forest, described by the following steps:

For b in $1, 2, ..., B$:

1. Sample n data points from the training set (with replacement), and call the features and labels X_b, Y_b.
2. Train a classification tree f_b on X_b and Y_b.

After these steps are completed, predictions on data points x can be made by taking the mode:

$$\hat{y} = \text{mode}(\{f_b(x)\}_{b=0}^{B})$$

Feature Bagging

In random forest classifiers, it is quite common to select a subset of features to train f_b on, randomly.

Considerations

Depending on the amount of data, around a few hundred to several thousand decision trees are used ($B \sim 10^3$). Additionally, for a classification problem with p features, it is typical to use $\sim \sqrt{p}$ features for each decision tree when it comes to feature bagging.

3.4.2 Hyperparameter Tuning

In training our model, we had the choice to tune some parameters. We started by tuning the parameter max_depth, which is the maximum depth of each decision tree f_b. Figure 9 shows the relation between the depth of the decision tree and the accuracy on validation data. It can be seen that the accuracy of the model levels out after max_depth passes 10. We must also be careful of not selecting a value too large for this parameter, as it can lead to overfitting.

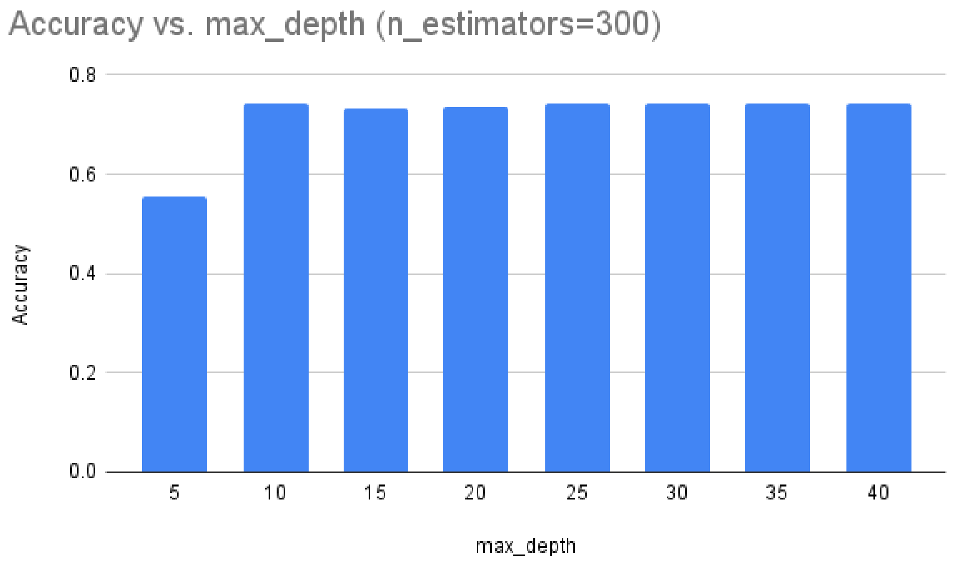


Figure 9: Tuning max_depth for Random Forest Classifier. Displays validation accuracy.

Another parameter we tuned was the number of estimaters, or n_estimators. This is essentially the value for B. Figure 10 shows the accuracy of the model for several values of B. It seems that $B = 200$ yields the best results on the validation data, but we chose 300 to be careful with overfitting.

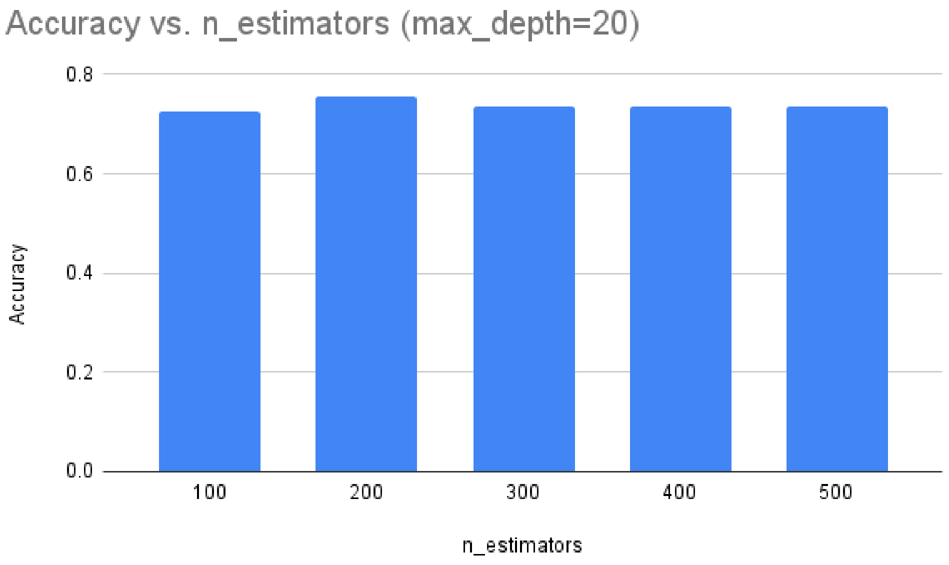


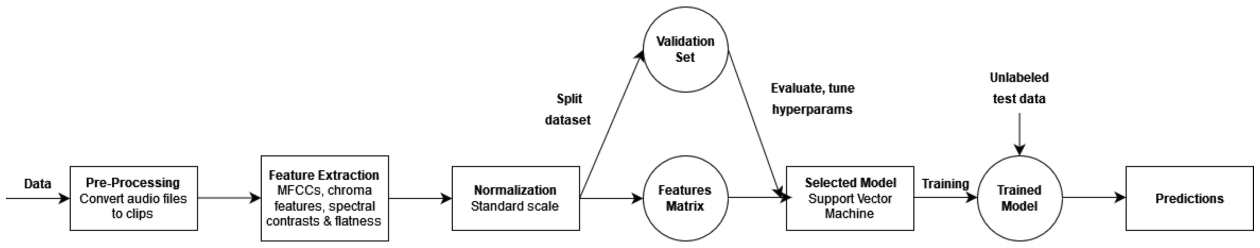
Figure 10: Tuning n_estimators for Random Forest Classifier. Displays validation accuracy.

4 Complete Pipeline

In summary, our pipeline was as follows:

- **Pre-Processing**: For each file in the training dataset, we split the file into ten 3-second clips (or twenty 1.5-second clips, as we did for our submission).
- **Feature Extraction**: For each clip, we extracted the chroma, MFCC, spectral contrast, and spectral flatness features. We transformed each of these matrices into vectors by taking their column-wise average. Finally, we created a data vector by concatenating the three vectors.
- **Model Training**: For more information about the model training process, consult the respective model's section.
- **Hyperparameter Tuning**: After our initial training attempts were relatively successful, we tuned features such as the number of epochs. Each model also posed unique challenges, such as the dropout probability for the neural network. Lastly, as discussed in the Feature Extraction section, we tweaked our extraction process (clipping or not clipping, changing the number of MFCC features, etc.) before settling on the data vector we did.

Below is a more detailed diagram of our pipeline:



5 Conclusion

The final model that we chose to use as our submission was the model (f) described in the SVM section. The audio files are shortened into 20 1.5-second clips to provide the model with more data to work with. MFCC, chroma, spectral contrast, and spectral flatness features are extracted from these shortened clips. None of the extracted features are removed. As described above, the SVM's hyperparameters that we use are 'C': 0.5, 'gamma': 'scale', 'kernel': 'rbf'. On Kaggle, the public test accuracy is 0.68, the highest of all our models. The graph below in Figure 11summarizes the highest accuracies of our 4 models.

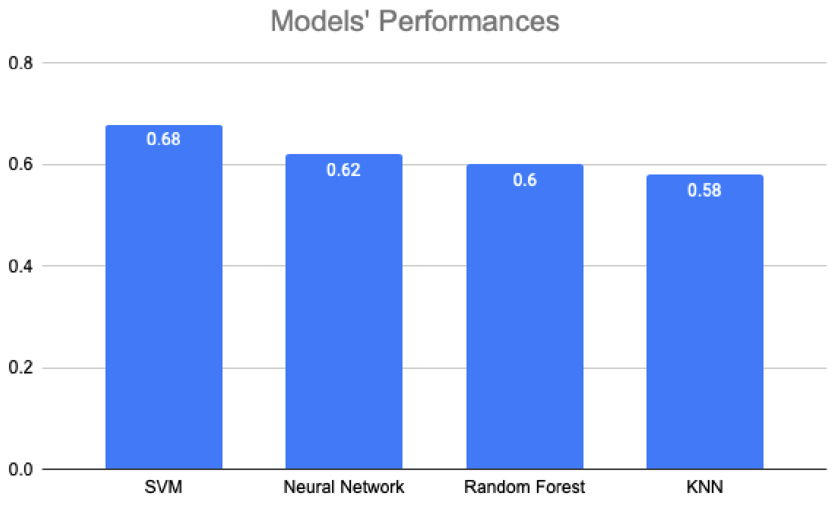


Figure 11: The highest public test accuracies of each of our 4 models

For the SVM model, the accuracies varied a lot based on the features extracted. As detailed in section 3.2.3, the SVM model was used to test out the different types of features extracted. SVM gave us the highest test accuracy score consistently on Kaggle, so we used it as our baseline for comparing the features extracted. The SVM model was not quite accurate enough when the audio clips are in their original 30-second form. At this point, it was reasonable for us to assume that the chosen RBF kernel and the regularization C value are not optimal, or that SVM is not a good model to start with. However, when the files are shortened into 10 3-second clips, the validation accuracy increased significantly, which means that with the chosen hyperparameters, SVM could actually perform well on the training data set. However, there is still a discrepancy between the validation and test accuries, which suggested ovefitting on the training data. The solution to mitigate this overfitting was to shorten the clips even more.

The combination of features we chose (MFCCs, Chroma, spectral contrast, and spectral flatness) represented all aspects of the audio. MFCC is good at detecting the variations in pitch because of its transformation into the frequency domain. These pitches are great for classifying genres because different genres have different primary pitches. Combined with the other three types of features, the models can give us a comprehensive overview of how the audio files are structured. The combination of features provides more information than any individual features alone. In our case, using only one type of features leads to underfitting: one type of features alone is not enough to classify the audio into genres.

SVM worked best in our case, as it is the method known to work best with high-dimensional data. CNN may require larger datasets to work as accurately, and random forest classifiers are still reliant on decision trees which may not be the most accurate classification method (as we mentioned, these decision trees tend to have problems with overfitting so we may need more data in this case as well).

A surprising aspect about the data science process is how the feature extraction process seems more important in improving our models' performances. In the beginning, I assumed that hyperparameter tuning was the most conducive towards a great model performance. However, after tuning the hyperparameters for a long time, I still could not improve the test accuracy by a lot. It turned out that the features extracted came into play a lot more than the hyperparameters. Using different tyeps of features (MFCCs alone or a combination of MFCCs, Chroma, and spectral contrast) made a significant difference in the model's performance.

The audio clips' duration was surprisingly also a main factor. Shortening the clips gives the training models a lot more data, which improved the test accuracy and reduced overfitting. In our case, no additional overfitting techniques were necessary to reduce the model's overfitting on the training data. We merely needed to shorten the audio clips by more so that clips can be used for feature extractions.

For future improvements, I would perhaps start working on the feature extractions first. In our group, only one person dedicated his time to extract features at first. I started working on the SVM model and tuning its hyperparameters. However, I later discovered that the feature extraction process is key to the models' performances.

Also, we did not explicitly try to reduce overfitting. In our final model, we shortened the audio clips, and somehow overfitting was alleviated by quite a lot compared to before. In future projects, I would apply more overfitting techniques like PCA or RFE directly to the model so that it would generalize even better. In this project, however, even without these overfitting reduction techniques, our model did not overfit on the public test data: its private test accuracy is still relatively high. But in more general cases, other techniques should be employed to mitigate overfitting.

6 Code

Refer to this link: `https://github.com/edwinlow2003/music-genre-classification/tree/main`

For code used to generate submission .csv files on Kaggle, refer to "Final Submission Code". "Testing Code" refers to code that we did not use to submit predictions, but may still be referred to by the report.

7 References

1. Random Forest (Wikipedia): `https://en.wikipedia.org/wiki/Random_forest`
2. MFCCs (Wikipedia): `https://en.wikipedia.org/wiki/Mel-frequency_cepstrum#:~:text=Mel%2Dfrequency%20cepstral%20coefficients%20(MFCCs,%2Da%2Dspectrum%22).`
3. Feature extraction with Librosa: `https://librosa.org/doc/0.10.1/feature.html`