Machine Learning-Based Audio Genre Classification

Group 3

Mateusz Ajose ma2226, Amelie Gonzalez - ag957, Oliver Longley - ol90,

Bernard Mendoza - BM535, Daniel Perry djp50, Conor Woollatt - cw734

GitLab repository

Project outline and goals

The goal of this project is to use and evaluate a range of AI techniques to detect the genre of a given piece of music. We are working with the GTZAN dataset, which contains 1,000 files of music evenly distributed across 10 genres. This dataset provides us with a benchmark to evaluate various methods of extracting and classifying data to ultimately create a robust and accurate classifier.

We will be using not only a variety of classifiers and algorithms, but also a variety of techniques to extract data features from the initial raw audio samples. The initial WAV files will be read in and converted to spectrograms, Mel spectrograms and onset envelopes among other formats. This will allow us to compare not only the models and algorithms themselves, but also to compare the value of different data features.

After we have finished evaluating all these considerations, we will then attempt to create an ensemble to combine multiple feature representations into one final model. This will allow us to combine the different strengths of each model to create an overall more accurate model. The ensemble may be created by combining multiple models with one of several techniques, chiefly stacking, bagging and boosting, or it may rely on simpler voting techniques to aggregate predictions.

Project structure and development

Once the project and dataset had been identified, we began to organise the development flow of the project, and to establish tasks we wished to achieve. We first began by researching a variety of various audio classification and data processing features that we could implement in our project. Different group members then were assigned different features and sections to begin researching so that we could divide the previously identified features to consider.

Once these features had been evaluated and understood, we were able to move forward with the practical implementation of each feature, of which the first one was data extraction and storage. The first phase of the project consisted of creating a variety of notebooks that took the raw audio inputs and extracted specific features using Librosa to be stored in an image representation.

Once this data had been produced, we were then able to begin creating classifiers from it. This was when the data began to be read in, resized and passed through the models and tested on their accuracy. Each model was tested in a variety of simple ways to judge the accuracy before choosing whether to proceed with or disregard the information.

Finally, once each base model had been created, we were able to begin optimising every constructing a final ensemble to combine our most successful methods and data features together.

Dataset and preprocessing

For this project we identified the GTZAN dataset, which is a music classification dataset made up of 1,000 WAV files split evenly between 10 genres (blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, rock). This provided us with a broad range of genres to classify and contained an equal amount of data per genre, which helped to prevent models from overfitting or underfitting specific genres.

Despite the well-structured and robustness of the data, we identified some minor issues during preprocessing. The first issue was the presence of a corrupt track (jazz.00054.wav) that was unable to be played and therefore couldn't be used in the creation of further data. While this was only one broken file in the genre, we considered removing either the file, or the genre entirely to prevent a genre being fitted differently to others. However, after reaching out to the creator of the dataset we were sent the missing file, and were able to resolve this problem, and complete the dataset with balanced classes.

The second issue we identified was that several files were slightly shorter, being 29 seconds instead of 30 seconds. This presented the possibility for different data files to be a different input shape, and therefore to cause issues in training. To address this issue, we ensured that all data was padded to a given length, and that any images were reshaped before use, therefore maintaining a consistent input shape for model training and testing.

Once these issues had been identified and resolved we were then able to proceed on to the steps of preprocessing and feature extraction. The library we chose to use for feature extraction was Librosa, a Python library for audio and music analysis. The library provided many tools to extract the various data features used in this project, and the outputs were able to be fed straight into the models used from Keras and sklearn.

With the dataset loaded, and Librosa ready for processing, we then began to split each audio file into 10 segments of equal length in order to increase the sample size. This process served two purposes: the first being to increase the number of files available for training and testing, and the second being to reduce the amount of information in each file. Keeping the two versions of the dataset (30 seconds, and 3 seconds) allowed us to test whether specific models performed better on shorter more specific data, or with more comprehensive data.

The main consideration we had to consider when splitting these files was that the separate parts of each file were kept together and either sent to the test, or to the train data. Due to the nature of music containing repeated patterns and motifs it was essential that they stayed together to prevent data leakage between the two sets. Had this occurred, it could have resulted in an overfitted model that learned a pattern from a song and was then tested on an almost identical section from the same track.

Evaluation methods

In order to evaluate the data, we will test the models predominantly on accuracy in order to simply assess the performance of a given model on given data. In order to test the accuracy, we will calculate the accuracy score by comparing the predicted classes to the actual classes of the data. This metric is simple and computationally cheap to calculate yet is able to provide a reliable evaluation of the model's performance. Alternatives to accuracy score include precision, recall, and F1 scores which all have their own situations where they provide greater value.

Precision is used to calculate how many true positive guesses were provided divided by the number of false true and positive guesses. It is more suitable for binary classification problems where there is an imbalanced amount of data per class, or false positives are more costly than false negatives. It is unsuitable for our project due to the multi-class data, and also due to the equal importance of each class and prediction.

Recall is similar to precision, but it works the other way around, where the score is calculated using true positives divided by true positives and false negatives. It is suitable for binary classification in a situation where false negatives are costly. Once again, these reasons make it unsuitable for our data.

The final accuracy metric we considered was F1 score, which calculates the balance between precision and recall. It therefore inherits the same limitations as the other two models, by being suited for binary classification with imbalanced classes.

In addition to accuracy, we will measure computational efficiency as a basic evaluator of the time taken to produce given results. While we haven't specified any requirements for computational efficiency, we will still be measuring it by benchmarking the models one by one running on the same system and hardware with minimal background processes. In addition to this, models will be trained and built using our own machines or the Jupyter server as required.

Additionally, we need to specify a method to use for splitting the data, with the two main options being a simple train-test split, and the second being k-fold cross validation. Train test split is applied to the data before any models are trained, and it splits the data into two sections, of which one section is exclusively used to train the models, and the other section is exclusively used to test the models. This is well suited to large datasets and complex models due to the simple nature of the split being very efficient, however it prevents a portion of the data from being used to train the models. On the other hand, k-fold validation provides the opposite result, where the dataset is split into k-equal size folds. The model is trained k times, where k-1 folds are used for training, and 1-fold is reserved for testing. This repeats with each fold being used in testing once, and for training k-1 times, and the results across each fold are averaged. It allows all the data to be used for training and testing. The downside is that it can be especially computationally expensive, especially for large datasets.

Our project contained both 1,000 long files, and 10,000 short files for each specific data feature, and therefore the dataset is at a large enough size where losing a portion of the data is not so significant, but that the training time would be vastly larger with k-fold. These reasons lead us to choose a simple train-test split to be the method used during the preparation of the data for our models.

Data formats

Waveform Analysis

The first method we attempted was to load and sample the WAV file 22,050 times per second to extract the amplitude of the audio signal. This sample rate is consistent with industry standards, with audio typically being recorded and saved at 44,100 Hz. The extracted data represents the amplitude of the wave, which indicates the loudness of the sound at a given time. We extracted 5 seconds of data from each file, then normalised and padded it to ensure consistency across the data. This gave us 110,250 samples per file with values between 1 and -1 to represent positive and negative heights. The data was then passed into a Support Vector Machine (SVM) which yielded an accuracy score of 0.14, a Dense Neural Network (DNN) which returned 0.145, and a Convolutional Neural Network (CNN) which was 0.175. Testing various configurations of the neural networks yielded minimal differences between iterations, and a confusion matrix of the DNN reveals a largely random set of predictions, with a tendency to overpredict hip-hop and pop. Analysing a confusion matrix for the CNN reveals a similar issue where classical, hip-hop and pop are the overwhelming majority of predictions, and 5 classes (blues, country, disco, metal and rock) are only predicted 3 times in total. Even though the accuracy of the CNN appears higher than the others it has only learned to predict 3 genres with any accuracy, and it has not learned the others whatsoever. A key reason for the failure of the SVM and DNN is because the data is generated is by sampling the wave over time, leading the order of the features to be highly relevant. The SVM and DNN are both unable to capture any of this temporal data. As a result, they likely rely only on the average volume of the song, treating each sample as an independent data point without considering surrounding context. By implementing a CNN with a convolutional layer, the model can attempt to extract patterns in the volume over time. While these results are poor, it is unsurprising given the limited scale of the data and lack of complex features to analyse. The base level of accuracy expected from a model simply guessing randomly would be 0.1, and therefore being 0.175 leads us to largely discount this data source from future consideration due to poor results.

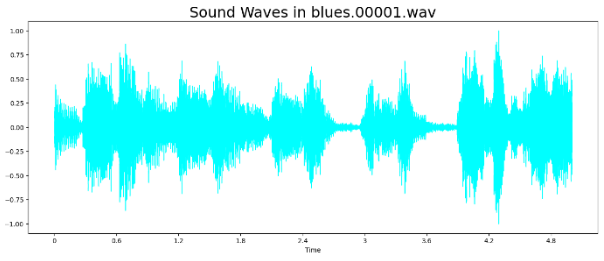


Figure 1- A waveform plotted against time

Mel Frequency Cepstral Coefficients (MFCCs)

The next method we used was analysing Mel-Frequency Cepstral Coefficients (MFCC) extracted from the WAV files. MFCCs are commonly used in speech and music analysis. They are extracted by converting the sound into the Mel scale which mimics human hearing and then extracting cepstral coefficients from the processed data. The raw audio is first converted into the frequency domain using the Discrete Fourier Transform (DFT), the Mel scale is applied to the resulting spectrogram followed by the logarithmic scale, and finally a Discrete Cosine Transform (DCT) extracts the cepstral coefficients. The extracted features represent the essential spectral characteristics of the audio as a small set of coefficients. Using Librosa, we were able to extract a specified number of these features for analysis, and in line with common practice we initially opted for 13 coefficients. The first 13 features are able to capture the most relevant information, and further increasing the number of features increases computation complexity without a significant increase in performance. The data was extracted and zero-padded to ensure a specified length, then saved as an array of size [1320,13] per song. The length of 1320 comes from a 30 second file at 22,050 samples being divided by Librosa's default hop length of 512 (661,500/512), then the data is by padded Librosa, and once more afterwards to ensure all samples match the largest one. As with the previous feature, the data was passed into an SVM, DNN and CNN to gauge an overall accuracy and to compare the different model's efficacy and suitability for the provided data. When presented with 13 features the models scored an accuracy of 0.43, 0.53 and 0.39 respectively, and these results shifted to 0.42, 0.35, 0.32 when presented with 20 features. These outcomes prove that 13 was an optimal number of features, due to the model's losing accuracy with the increase. When run with 13 features the models took 0:59, 0:55 and 3:35 to run, and then when trained on 20 features they took 1:26, 1:12 and 5:14. This resulted in an average duration increase of ~40% for a less accurate classifier. The results from this set of features were far more promising than the previous method, with all models around an accuracy of 0.4-0.5. Reviewing confusion matrices generated for all models reveals that classical and metal were classified especially well, while predictions for the other genres were more broadly distributed. The strong performance of metal and classical may be due to their lower emphasis on the vocals and more distinct instrumental styles compared to other genres. Additionally, these features once again contain data that is encoded over time, that the SVM and DNN were unable to process, yet they managed to achieve reasonable results regardless. This evaluation has indicated that MFCCs are a valuable feature to proceed with analysing.

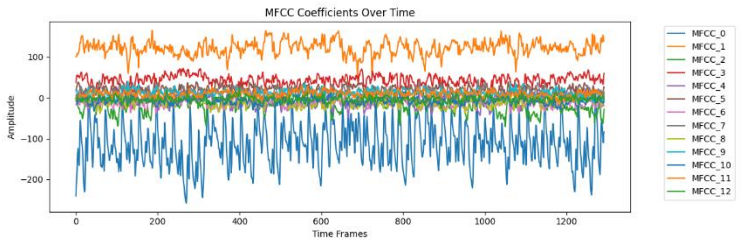


Figure 2 - A visualisation of each Mel feature across time

Spectrograms

The next data method we chose to explore was Spectrograms generated from the WAV files. Spectrograms are a representation of the frequency of a signal over time, where the time and frequency are represented by the x- and y-axis, and the intensity is represented by pixel colour. They convert a two-dimensional WAV into a three-dimensional image that visualises frequency changes over time. They are generated by applying the Short-Time Fourier Transform (STFT) to segments of an audio signal and the results are stacked to show frequency power. The transformation to generate the files uses a parameter called hop length, which specifies the distance the frame will shift between each sample. A smaller hop length moves the frame further and generates a smoother spectrogram, but at the cost of computational efficiency. We tested two iterations of this, using Librosa's default hop length of 512, and a shorter length of 256, and the images were saved at full resolution. Once the data had been extracted it was passed into a CNN to attempt to identify features and create a classifier. The large volume of data being passed into the model caused it to be more computationally expensive and taking longer than previous data features. The input to this model was 10,000 images that had to be resized to 256x256, and then 8000 were trained on over 25 epochs. The result of this model was an accuracy of 0.672 for the 256 spectrograms, and the 512 spectrograms scored 0.726. This result is somewhat surprising, given that the less detailed images produced a lower accuracy result than the defaults. The time taken to run these models was 90 minutes for the 256 images, and 83 minutes for the 512 images. This similarity is to be expected due to an equal number of input images being reshaped to the same size and being passed into a CNN with the same architecture. While it seems strange that the higher density images produced lower accuracy results, it may be explained by the way we generated and handled the images. When the images were generated, there was no file size or resolution specified to be saved at, this led to the 256 and 512 images being different. While all the images were saved as 600x1200 pixels, the 256 images were approximately 50% larger (saving at 360KB per fie, vs 210KB per file). When viewing the full file at maximum resolution the higher resolution could allow greater detail to be retained. However, since the files are resized when they are loaded back in this additional data is lost. This likely leads to a greater loss of data for the 256 files, and therefore a loss in accuracy when run through the CNN. Testing was attempted with images reshaped to 512x512 when loading them in, but the additional computational complexity made it unfeasible for the scope of our project. This complexity came in the form of both the time taken to train the model, but also the memory taken to have every image loaded in memory. In addition to training the models separately, we also experimented with training a single CNN with both images passed in simultaneously as separate channels. This multi-channel approach would introduce the ability to learn from both images at the same time, and to deduce deeper patterns across hop lengths. The model took 10 minutes to train, and it ended up with an accuracy of 0.674 and a confusion matrix revealed that it was had fitted well across all the genres, but metal and country were especially well fitted comparatively. Overall, these results provided us with the most accurate results yet even at the cost of computation time.

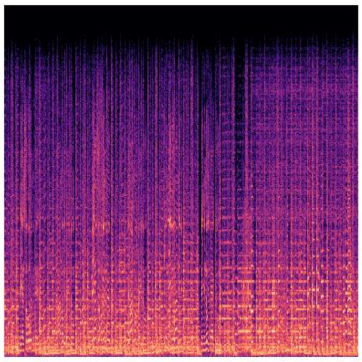


Figure 3 - A Spectrogram

Mel-Spectrograms

Mel Spectrograms, much like regular spectrograms, are another form of representing the data both with time and frequency, but in relation to the Mel scale. They are generated much like the regular spectrograms, except they apply Mel filters in place of a linear frequency scale. These images compress the higher frequencies and emphasise the lower frequencies that are more relevant to human hearing. We again used Librosa's tools to generate the Mel spectrograms by specifying the hop length, the number of Mels, and the sample rate. The number of Mels defines how many frequency bands are used, and therefore how much data is captured in the image. Typical number of Mels range from 40 for speech analysis, to 128 for genre classification, through to 256 for instrument identification. We decided to test three numbers of Mels, and we generated images using 32, 128, and 512 Mels. Like the regular spectrograms, the images were generated in the same dimensions, but with different file sizes. The 32 Mel images generated to be roughly 20KB each, the 128 Mel images generated to be 60KB each, and the 512 Mel images were 200KB. Once again, the data was fed into a CNN comprised of several 2D convolutional layers with max pooling, then flattened and passed into a series of fully connected dense layers. The models took 79 minutes for 32 Mels, 81 minutes for 128 Mels, and 83 minutes for 512 Mels. This showed that increasing the number of Mels and the file size didn't substantially increase computation time, since all images are resized to the same dimensions. The models then returned accuracies of 0.637, 0.674, and 0.692 respectively. These results show that an increased number of Mels leads to a slightly higher accuracy at the cost of file size and a fractional increase in computation time. These results are largely unsurprising even when compared to the genre classification standard of 128. These standards are likely based off of the number needed to extract the majority of the relevant features. Additionally, we created a multi-channel CNN which read in all 3 images at once to attempt a comparison across all sizes. While each image was similar to the others, this model would be able to compare patterns from all 3 densities at once and therefore be able to extract patterns not visible in just one image. The model ran in 83 minutes and had an accuracy score of 0.687 showing that there was no real improvement from stacking the models. This is likely due to the fact that accuracy increased with the number of Mels, and therefore the higher Mel filters preserved more data. It is likely that the accuracy was averaged downwards by the less useful 32 Mel images, and that there are no different patterns in lower Mel images. Overall, Mel spectrograms have produced a good accuracy largely on par with the spectrograms, therefore suggesting the value in considering both features going forwards.

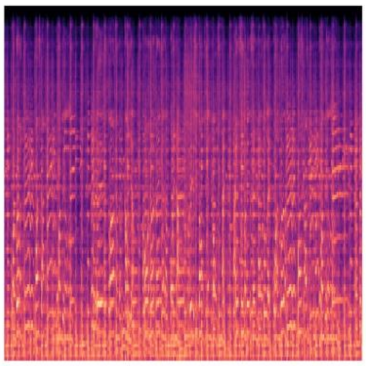


Figure 4 - A Mel-Spectrogram

Chroma features

The next method we analysed was a less typical method of extracting and visualising data extracting chroma features and storing them in chromagrams. The data extracted and stored in a chromagram represents the intensity of musical pitch classes across time showing the harmonic content of audio. It is created by projecting the frequency spectrum onto the 12 notes of the chromatic scale, and it is therefore suited for analysing chords and melodies. It is worth noting that chroma features are optimised for western music that relies on the chromatic scale, however for our project this wasn't a problem due to our genres all using the chromatic scale. We used Librosa to extract these features and store them with matplotlib diagrams across a range of bins from 12, 24 and 36. We then planned to feed these images into a CNN like the other images, but unfortunately the accuracy never increased above the base level of 0.1-0.15. It is possible that the model was structured incorrectly, or that the data was being analysed in an unsuitable way as chromagrams fall short in the more complex melodies where the pitch classes being recorded begin to muddle together. These images could not be recorded with the ideal resolution as chromagrams are typically rather long images that need to span over shorter time intervals to provide meaningful information. This increase in resolution, however, would also increase the amount of redundant data and possibly hinder results even further, resulting in us disregarding the data to progress down more promising paths.

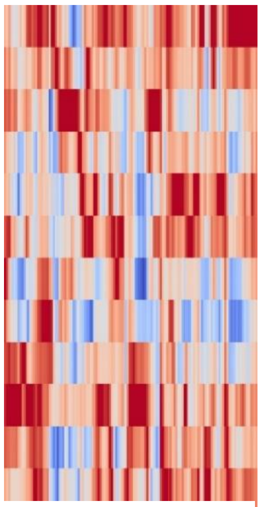


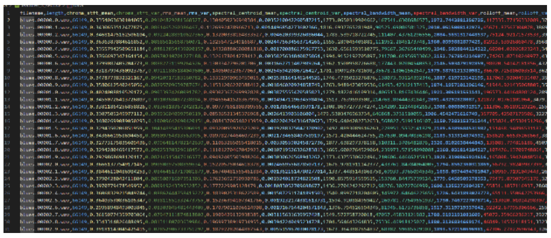
Figure 5 - A chromagram

Onset

Onset images were another technique we investigated as a potential data point. Onset images and heatmaps were a type of data that we generated using onset information from Librosa. Onset information is a way of representing when notes begin in audio, and it is generated by detecting sudden energy changes in the audio. Similarly to chroma features the CNN was unable to make significant progress, and accuracy stalled at 0.38. While a confusion matrix did reveal that there was a clear attempt to classify the genres, it also showed a significant random distribution. The classes that onset heatmaps were able to predict most accurately were disco and metal. This is likely due to the punchy nature of the genres, with disco featuring bouncy beats, and with metal emphasising percussion. Despite the clear attempt at classification, it is clear that there was not enough information stored in the heatmaps to produce meaningful results. Like with chromagrams this led us to discard the features as not individually valuable enough.

Feature Extraction

Once all the image based data features had been extracted we explored an alternative method of extracting and saving the features. Instead of extracting and storing the features in images, we used tools built into Librosa to extract the data and save it into a CSV format. The features we included contained, the mean and variance of chroma features, spectral features, Mel features as well as the mean and rate of energy sign change. Additionally, we extracted the harmonic and percussive strengths, and finally the tempo. This data was stored in a csv with the filename, track length, features and the genre, and we repeated this process for both 3, and 30 second clips. Since the data was extracted as tabular features without temporal or spatial information, the data was suitable to be passed into a larger variety of models than the CNNs which were the only appropriate way to extract and analyse data from images. In addition to the previously used CNN, DNN, and SVM models, we introduced two new classifiers: Random Forest (RFC) and Extreme Gradient Boosting (XGB). Although the extracted features do not explicitly contain any temporal data a CNN was still incorporated to test whether any data is present in the order of the Mel features. The results of running the data through these models with the 30 second features were 0.725, 0.495, 0.700, 0.760, 0.785. Then the 3 second features file was used with the same models which scored 0.910, 0.904, 0.860, 0.883, 0.918 respectively. These results were exceptionally good for independent models without any enhancements or ensembles applied. In order to compare the cohesion of each model, the number of unanimous matches between the models was calculated. Out of all 5 models tested on 1,998 rows of data it was revealed that there were 1,551 instances where every model predicted the same result, and that overall, every model matched the predictions of any other between 81% and 86%. Despite their high accuracy, the models also trained quickly, with the 3 second models training in 1:17s, 0:32s, 0:07s, 0:28s and 0:05s. These results proved the extracted features to not only be more accurate and more efficient, but they also allowed a greater variety of models for inclusion in a final model.



```
filename,length,chroma_stft_mean,chroma_stft_var,rms_mean,rms_var,spectral_centroid_mean,spectral_centroid_var,spectral_bandwidth_mean,spectral_bandwidth_var,rolloff_mean,rolloff_va
[illegible]
```

Figure 6 - A section of the features CSV

Principal Component Analysis

Now that we had generated tabular data, the next method we attempted was Principal Component Analysis (PCA), which reduces the dimensions of data by finding orthogonal axes (principal components) that maximise variance. They are calculated by normalising the data, computing the covariance matrix and performing eigenvalue decomposition, and a specified number of components are generated. The most common number of values is either 2 or 3, where they cumulatively capture 85% and 95% of the variance respectively. We opted to extract and test both 2, and 3 principal components from the data to compare if there was a difference in computational cost, or accuracy of these methods. In order to visualise this data, we plotted a diagram using MatplotLib and calculated the centroids of each genre to view the distribution across the Principal Components.

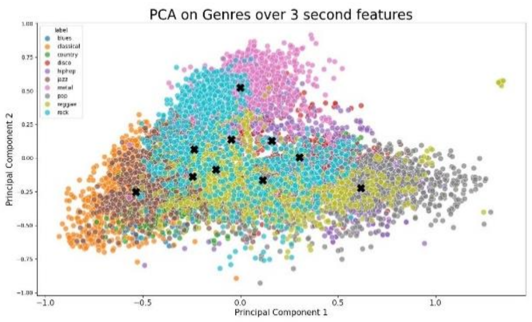


Figure 7 - A visualisation of the genres by PCA

Using the previously generated features CSV we were able to reuse this data to extract new features from. Once this data had been saved into the CSV, we were ready to load it back into the project and process it before passing it to the models to learn. We loaded the data into a pandas dataframe and split the features and genres into X and y, we then scaled the X data to normalise it and encoded the genres as numeric vales. Scaling the data ensures that all the features are evenly represented, and that no larger features sway the principal components. The components were then calculated with SKlearn and saved to a new set of test and train data comprised of just the 2 or 3 features which are each represented by either a positive or a negative number. Once these features were extracted, we then began to feed them into the same models used previously of a CNN, DNN, SVM and XGB which were all tested on both the 3 and 30 second feature files. When extracting two features over 30 seconds, the models scored 0.110, 0.155, 0.120, and 0.150 respectively. Then when presented with the 3 second files they scored 0.217, 0.214, 0.215 and 0.178. Then, when extracting 3 features on the 30 second files the models all performed similarly with results of 0.15, 0.135, 0.145 and 0.130 respectively. And on the 3 second files they scored accuracies of 0.268, 0.260, 0.250, and 0.247. These results clearly showed that the 3 second features were significantly more accurate, but also that extracting 3 features was more accurate also. Due to the small amount of data, these models are able to run almost instantly, all running in under 3 seconds each. While these features provide no value on their own, their low computational requirements and simplicity make them more a valuable feature to incorporate in further models and data.

Ensemble techniques

Now that each data feature had been explored, and baseline accuracies of each model had been recorded, we were able to begin looking into creating an ensemble of multiple models.

Different models trained on different all have different strengths and weaknesses, and therefore one individual model is unlikely to reach as high of an accuracy as multiple models. Not only will combining models allow us to combine the strengths of each model, but it will allow us to average out and ignore the weaknesses and overfitting of any one particular model. In order to combine the outputs of multiple models, ensemble techniques can be used to combine multiple models and their predictions.

Stacking involves training multiple base level models and combining them into one meta model. This allows multiple types of models that may better recognise specific patterns better than others to be combined, but it comes at the downside of a more computationally expensive model. In our project one of the base models, we used Support Vector Machines (SVMs), which are an algorithm that implement stacking.

Bagging is a similar technique that works by splitting the dataset into multiple subsets, and training individual models to then be combined at the end into a final model that will predict using majority voting of each sub model. Bagging aims to reduce overfitting by giving each sub model on a different set of data, however it can struggle without tabular data. In our project one of the base models, we used Random Forests which implement stacking to create a classifier.

Boosting finally involves creating multiple models sequentially, whereby each model attempts to correct the errors made by the previous model and the final prediction is determined by weighted voting. It is able to avoid bias by combining multiple weaker models that have not been able to overfit. However, it is prone to struggle with outlying data as each model has less scale. In our project, one of the most powerful models we used was Extreme Gradient Boosting (XGB), which uses boosting to create a classifier.

An alternative to these complex models is voting methods such as majority or weighted voting, which take the predictions from multiple models and produce an overall prediction. Majority voting involves each model producing a prediction and the class with the most votes wins. Meanwhile, weighted voting assigns each model a weight based on testing accuracy, and the predictions are calculated using the weight assigned to each model. These two voting methods can also be either soft or hard, where soft voting requires a simple prediction, and hard voting requires the models to generate a probability that is used to weight certain predictions higher than others.

Final ensemble

For our final ensemble we tested using both the features CSV and also using some of the image formats through a CNN in order to compare the effectiveness of various configurations and combinations. The first step we took was to calculate the Principal Components from the provided features, and then they were appended to the data frame to be included in the tabular data. This data was fed back into the same models, being CNN, DNN, SVM, RFC and XGB. At the beginning of the ensemble creation the models were at individual accuracies of 0.898, 0.814, 0.756, 0.876, 0.918 respectively, which were all significantly accurately, but the end goal was to create an ensemble that took the best features and representations of features to produce an output. The first step taken towards creating a more powerful ensemble was to optimise every model to extract the maximum accuracy possible and to aim to bring them to a similar level. For the CNN and DNN the ese optimisations revolved around tweaking the number of layers, the neurons per layer, the number of epochs, and the batch size. Additional improvements in the form of callbacks during training to allow the models to adjust as they trained. These models ended up hitting accuracies of 0.922 and 0.908. This increase was especially large for the DNN which was previously very simple and under optimised. The next model was SVM which had performed well overall but underperformed compared to the other models. The first step was to scale the data being fed into the models so that it was normalised between -1 to 1. We then experimented with tuning the hyperparameters to optimise the model more, and after testing these the model reached 0.926 which represented a substantial increase of over 0.15. We then moved to optimise the RFC by considering whether a maximum depth should be specified, but it was determined that the model was not overfitting and was left unmodified at 0.876 which was on par with the SVM. Finally, the XGB model was reviewed, but was again determined to be well optimised by default for our needs. The final accuracies for the models ended up at 0.906, 0.908, 0.926, 0.874 and 0.919, which represented an average increase of over 0.05.

After each model had been optimised, we then began to implement a soft and hard voting classifier. The soft voting model required each model to produce a list of class predictions with a probability assigned to each. These probabilities were then added and normalised to create an average prediction per class, which was then used to produce a final prediction. This model included all 5 of the individual models, and it produced an accuracy of 0.943 which was by far the highest result to date. The hard voting model worked much the same, except each model produced only 1 prediction and the most voted prediction was final. This model produced an almost identical result of 0.942

We then experimented with a weighted voting model, where the accuracy of every model is summed and normalised to produce a weight that is then applied to its probability predictions. Then, just like a soft voting model the highest predicted class is selected. This model produced an accuracy of 0.938 making it the lowest ensemble technique. This may be due to the

The final model we created was a stacked ensemble which combined every individual model with its predictions into a new stacked train dataset and then attempts to learn from that. The meta model generated by our stack gave an accuracy of 0.950 which was overall the highest accuracy generated by this project. This accuracy was a big milestone to hit, as it

In addition to these we attempted to incorporate one of the image-reading CNNs into the ensemble to combine the accuracy. However, due to the low accuracy of the CNNs averaging around 0.65-0.70, we anticipated that the lower accuracy CNN would likely drag the ensemble's accuracy down a little. To test this theory, we implemented the highest scoring Spectrogram reading CNN into the soft voting ensemble, where the file path to each clip was derived from the filename stored in the features CSV. The features csv was then split, and the filenames stayed with the correct features row. Once again, the models were trained individually, with each flat model trained on the same train data, and the CNN being trained on a new array generated by loading each file name in order. Once these models were created the ensemble then scored an accuracy of 0.87. Which, while an individually impressive score, was unfortunately much lower than the accuracy of just the tabular data.

Reflections

Reflecting on the decisions made across the project we were able to identify several parts of our approach that we would have changed. The vast amount of data generated was a challenge when working with this project, as every set of testing involving CNNs required 10,000-30,000 images to be input, resized and trained for long periods of time. In order to mitigate these challenges, we would have specified a size for the images to be saved as when generated. An additional issue with the volume of data is that each group member had to generate the images locally, this is because by the end of the project we had created over 17GB of data.

Additionally, we wrote off datapoints like chromagrams and onsets a non-valuable feature, however with the correct processing these features would have likely been able to generate somewhat accurate predictions.

Conclusion

The project successfully achieved its primary goal, by progressing from a waveform analysis at 17.5% accuracy through to a 95% accurate ensemble based on varied features. We evaluated a variety different features to be extracted from the data, and we extracted, processed and saved these features in a number of different formats. We then created individual classifiers to determine the accuracy and feasibility of interpreting the genre from these data points. Once these models had been trained individually, we were able to combine them in an ensemble to average out the predictions of each model and create a final most accurate classifier. The project began with a waveform analysis model that was close to random accuracy and progressed through to an ensemble combining features extracted from the data combined with Principal Components, that was passed into a series of stacked models that produced an accuracy of 95%.