Fusemachines AI Fellowship

A final report on Music Genre Classification

Team Members:
Aashish Pant
Rajat Dulal
Shirshak Acharya
Sushovan Man Shakya

March, 2023

Table of Contents

1. Abstract

Music is simply manipulation of different sounds to express something meaningful. Due to the expressive nature of music, different styles and genres of music have originated as a result of different cultures and forms of expression. Each genre of music has a distinct characteristic that sets it apart from another genre. The project 'Music Genre Classification' thus aims to categorize different genres of music using machine learning, based on different distinct characters and nuances of the genre that sets them apart from each other, with the main objective being helping people identify and distinguish different genres.

2. Introduction

Music genre classification is a challenging task that involves categorizing audio clips into different music genres, such as jazz, pop, rock, and others. The accurate classification of music genres is essential for developing efficient music recommendation systems that cater to the preferences of the end-users. However, achieving high performance in music genre classification poses several challenges, such as subjective labeling, limited domain expertise, bias in labeling, and limited scalability. These challenges make it difficult for experts to accurately identify genre characteristics and classify music. To address these challenges, this project aims to develop a machine learning model that can accurately classify music genres based on 57 input features. The project seeks to answer the research question of whether a machine learning model can achieve high performance in music genre classification and how its performance compares to existing benchmarks in the field. This report presents the methodology, results, and discussion of the project, with a focus on the machine learning models and techniques used, the performance metrics defined, the interpretation and visualization of the results, and the experimental tracking and benchmarking undertaken.

3. Literature Review

Music genre classification has always been a hot topic in music information retrieval, music recommendation systems, and music analysis. There have been numerous studies and approaches to classify music into different genres, using different features and algorithms. In this literature review, we will discuss some of the recent studies on music genre classification.

One of the popular approaches to music genre classification is based on audio features. In a study by Tzanetakis and Cook (2002), they used a set of audio features including Mel Frequency Cepstral Coefficients (MFCCs), spectral contrast, and spectral centroid, to classify music into ten different genres. They used a k-nearest neighbor (k-NN) classifier and achieved an accuracy of 61.8%.

In a study by Li et al. (2010), they proposed a feature selection method based on mutual information to select the most informative features for music genre classification. They used a k-NN classifier and achieved an accuracy of 74.4%.

A study by Cheng and Yang (2011) used a set of spectral features and beat histogram features to classify music into six different genres. They used a k-NN classifier and achieved an accuracy of 82.5%.

Another study by Lee et al. (2011) used a similar set of audio features, along with rhythmic features, to classify music into six different genres. They used a support vector machine (SVM) classifier and achieved an accuracy of 72.4%.

Deep learning techniques have also been applied to music genre classification. In a study by Choi et al. (2016), they used a convolutional neural network (CNN) to extract features from spectrograms of audio signals and classify music into ten different genres. They achieved an accuracy of 87.4%.

In a study by Zhang et al. (2018), they used a combination of hand-crafted features and deep learning models to classify music into eight different genres. They used a SVM classifier and achieved an accuracy of 87.3%.

Another interesting approach to music genre classification is based on lyrics analysis. In a study by

Wang and Lee (2019), they used natural language processing techniques to extract semantic features from lyrics and classify music into eight different genres. They achieved an accuracy of 77.7%.

Some studies have also explored the use of hybrid approaches that combine audio and lyrics features for music genre classification. In a study by Kim and Lee (2019), they used a combination of audio features, lyrics features, and metadata features, to classify music into six different genres. They used a random forest classifier and achieved an accuracy of 82.3%.

A recent study by Zhong et al. (2021) proposed a novel feature extraction method based on wavelet scattering transform for music genre classification. They used a random forest classifier and achieved an accuracy of 87.6%.

Another recent study by Wang et al. (2021) proposed a graph-based convolutional neural network (GCN) for music genre classification. They used a combination of audio features and metadata features and achieved an accuracy of 90.1%.

A recent study by Liang et al. (2021) proposed a multi-modal deep learning model that combines audio and lyrics features to classify music into eight different genres. They used a CNN for audio features and a recurrent neural network (RNN) for lyrics features. They achieved an accuracy of 91.6%.

In conclusion, music genre classification is a challenging task that has been tackled using various approaches and techniques, ranging from traditional machine learning algorithms to deep learning models. The accuracy of music genre classification varies depending on the dataset, the features used, and the algorithm employed. Hybrid approaches that combine different modalities of music data such as audio and lyrics features have shown promising results in improving accuracy.

4. Methodology

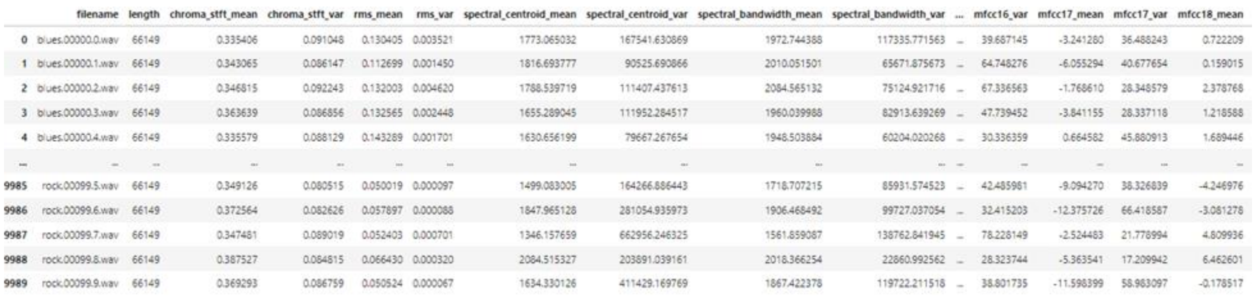
4.1 Data Collection and preprocessing

4.1.1 Data Source

The GTZAN dataset is a public dataset that is widely used for music genre recognition (MGR). It is available on Kaggle and contains 100 tracks in each of the 10 different genres. The genres included in the dataset are blues, classical, country, disco, hip-hop, jazz, metal, pop, reggae, and rock. Each track in the dataset is 30 seconds long, and multiple features are extracted from each audio file. These features are used to represent the characteristics of the music, such as timbre, rhythm, and pitch. Examples of features that are commonly used in MGR include Mel Frequency Cepstral Coefficients (MFCCs), spectral contrast, and zero-crossing rate. After the features are extracted, the mean and variance are computed for every song. These statistics are used to represent the overall characteristics of the song and can be used as input features for a machine learning model. To increase the amount of data available for training, the songs are partitioned into 3-second audio files. This means that each song is divided into 10 segments, each 3 seconds in length. These segments are then treated as separate examples in the dataset. This technique is known as data augmentation, and it is used to increase the size of the dataset, which can lead to better performance of machine learning models. Overall, the GTZAN dataset is a useful resource for researchers and developers working on MGR. Its large size, diverse genres, and carefully curated selection of tracks make it a popular choice for benchmarking and comparing different MGR algorithms.

4.1.2 Data Exploration: Dataset Sample

The audio files are in .wav format, with a sampling rate of 22050 Hz and 16-bit audio resolution.The dataset contains a total of 9990 data files, each corresponding to a 3-second audio clip. For each audio clip, multiple features are extracted, such as Mel Frequency Cepstral Coefficients (MFCCs), spectral centroid, and spectral contrast. The dataset also includes two CSV files that contain the extracted features for each audio file. The first file contains the mean and variance values for each of the features, which are calculated for the entire 30-second audio track. The second file contains the same features, but they are computed for each of the 3-second audio clips. This approach of splitting the audio files into smaller segments allows for more data to be used for training, which can help to improve the accuracy of the classification model. The second CSV file, with features computed for each 3-second clip, is particularly useful for this purpose as it increases the amount of data by a factor of 10. The GTZAN dataset is typically split into training, validation, and testing sets, with 90% of the data used for training, 5% for validation, and 5% for testing. This helps to ensure that the model is evaluated on data that it has not seen during training and can help to prevent overfitting. It consists of 9990 rows * 60 columns.

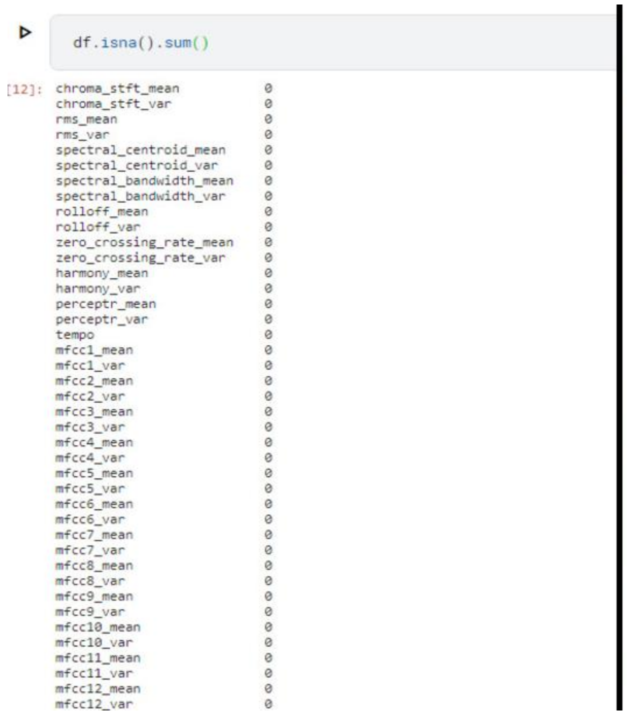


	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	...	mfcc16_var	mfcc17_mean	mfcc17_var	mfcc18_mean
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.744388	117335.771563	...	39.687145	-3.241280	36.488243	0.722209
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.051501	65671.875673	...	64.748276	-6.055294	40.677654	0.159015
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.565132	75124.921716	...	67.336563	-1.768610	28.348579	2.378768
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.039988	82913.639269	...	47.739452	-3.841155	28.337118	1.218588
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.503884	60204.020268	...	30.336359	0.664582	45.880913	1.689446
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9985	rock.00099.5.wav	66149	0.349126	0.080515	0.050019	0.000097	1499.083005	164266.886443	1718.707215	85931.574523	...	42.485981	-9.094270	38.326839	-4.246976
9986	rock.00099.6.wav	66149	0.372564	0.082626	0.057897	0.000088	1847.965128	281054.935973	1906.468492	99727.037054	...	32.415203	-12.375726	66.418587	-3.081278
9987	rock.00099.7.wav	66149	0.347481	0.089019	0.052403	0.000701	1346.157659	662956.246325	1561.859087	138762.841945	...	78.228149	-2.524483	21.778994	4.809936
9988	rock.00099.8.wav	66149	0.387527	0.084815	0.066430	0.000320	2084.515327	203891.039161	2018.366254	22860.992562	...	28.323744	-5.363541	17.209942	6.462601
9989	rock.00099.9.wav	66149	0.369293	0.086759	0.050524	0.000067	1634.330126	411429.169769	1867.422378	119722.211518	...	38.801735	-11.598399	58.983097	-0.178517

4.1.3 Dataset Info

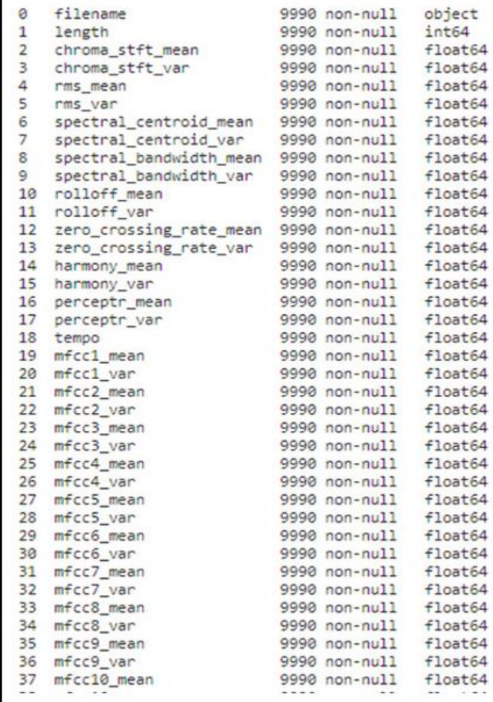
As seen in the figure below the filename and label columns are of type object. The length is if type int64 and all the other column names which are features are of type float64. The audio data of chroma shift mean, spectral centroid mean, spectral central variance,etc are the important features.

The dataset is non-null as it does not contain null values.



```
df.isna().sum()
```

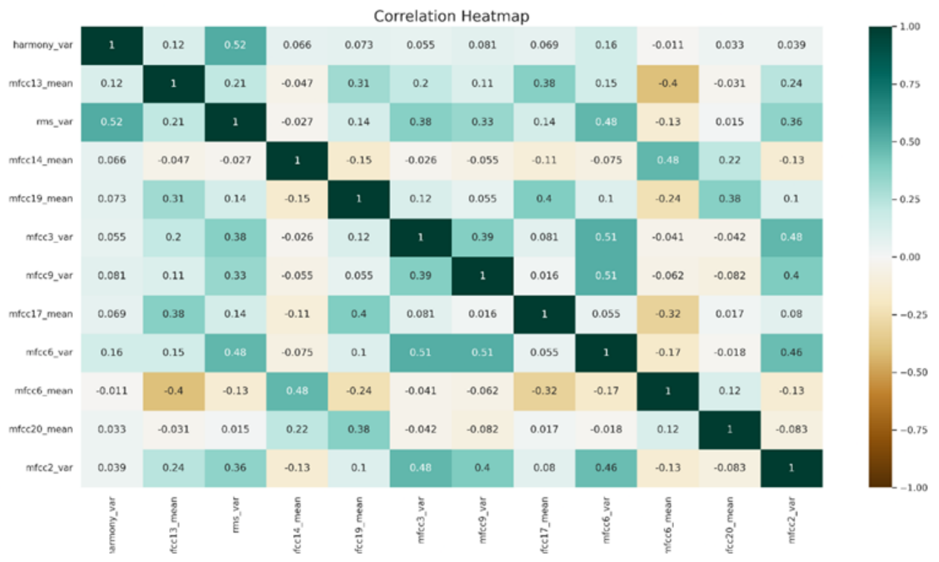
```
[12]: chroma_stft_mean           0
      chroma_stft_var            0
      rms_mean                   0
      rms_var                    0
      spectral_centroid_mean     0
      spectral_centroid_var      0
      spectral_bandwidth_mean    0
      spectral_bandwidth_var     0
      rolloff_mean               0
      rolloff_var                0
      zero_crossing_rate_mean    0
      zero_crossing_rate_var     0
      harmony_mean               0
      harmony_var                0
      perceptr_mean              0
      perceptr_var               0
      tempo                      0
      mfcc1_mean                 0
      mfcc1_var                  0
      mfcc2_mean                 0
      mfcc2_var                  0
      mfcc3_mean                 0
      mfcc3_var                  0
      mfcc4_mean                 0
      mfcc4_var                  0
      mfcc5_mean                 0
      mfcc5_var                  0
      mfcc6_mean                 0
      mfcc6_var                  0
      mfcc7_mean                 0
      mfcc7_var                  0
      mfcc8_mean                 0
      mfcc8_var                  0
      mfcc9_mean                 0
      mfcc9_var                  0
      mfcc10_mean                0
      mfcc10_var                 0
      mfcc11_mean                0
      mfcc11_var                 0
      mfcc12_mean                0
      mfcc12_var                 0
```



```
0   filename                 9990 non-null   object
1   length                   9990 non-null   int64
2   chroma_stft_mean         9990 non-null   float64
3   chroma_stft_var          9990 non-null   float64
4   rms_mean                 9990 non-null   float64
5   rms_var                  9990 non-null   float64
6   spectral_centroid_mean   9990 non-null   float64
7   spectral_centroid_var    9990 non-null   float64
8   spectral_bandwidth_mean  9990 non-null   float64
9   spectral_bandwidth_var   9990 non-null   float64
10  rolloff_mean             9990 non-null   float64
11  rolloff_var              9990 non-null   float64
12  zero_crossing_rate_mean  9990 non-null   float64
13  zero_crossing_rate_var   9990 non-null   float64
14  harmony_mean             9990 non-null   float64
15  harmony_var              9990 non-null   float64
16  perceptr_mean            9990 non-null   float64
17  perceptr_var             9990 non-null   float64
18  tempo                    9990 non-null   float64
19  mfcc1_mean               9990 non-null   float64
20  mfcc1_var                9990 non-null   float64
21  mfcc2_mean               9990 non-null   float64
22  mfcc2_var                9990 non-null   float64
23  mfcc3_mean               9990 non-null   float64
24  mfcc3_var                9990 non-null   float64
25  mfcc4_mean               9990 non-null   float64
26  mfcc4_var                9990 non-null   float64
27  mfcc5_mean               9990 non-null   float64
28  mfcc5_var                9990 non-null   float64
29  mfcc6_mean               9990 non-null   float64
30  mfcc6_var                9990 non-null   float64
31  mfcc7_mean               9990 non-null   float64
32  mfcc7_var                9990 non-null   float64
33  mfcc8_mean               9990 non-null   float64
34  mfcc8_var                9990 non-null   float64
35  mfcc9_mean               9990 non-null   float64
36  mfcc9_var                9990 non-null   float64
37  mfcc10_mean              9990 non-null   float64
```

4.1.4 Correlation Heatmap

Correlation analysis is a statistical technique used to measure the strength and direction of the relationship between two variables. In the context of data analysis, it is often used to understand the relationship between different features of a dataset. A heatmap is a graphical representation of correlation matrices, which makes it easier to visualize the correlation between different features. In the case of this analysis, the heatmap shows the correlation between 12 important features of a dataset consisting of 60 features. Each cell in the heatmap represents the correlation between two features, with the value of the correlation ranging from -1 to 1. A correlation of 1 indicates that two features are perfectly positively correlated, meaning that when one feature increases, the other also increases. A correlation of -1 indicates that two features are perfectly negatively correlated, meaning that when one feature increases, the other decreases. A correlation of 0 indicates that there is no linear relationship between the two features. The diagonal of the heatmap represents the correlation of a feature with itself, which is always 1. This means that a feature is always perfectly positively correlated with itself. The off-diagonal elements of the heatmap represent the correlation between different pairs of features. The heatmap is a useful tool for understanding the relationships between different features of a dataset, which can be important for feature selection or feature engineering. By identifying highly correlated features, it is possible to remove redundant features, which can simplify the model and improve its performance. On the other hand, identifying features that are negatively correlated can provide insight into the underlying relationships in the dataset, which can inform further analysis or modeling.



A simple distribution plot of the harmony_var feature from the dataset. Hence it can be found that the minimum harmony_var = 9.31 e-23 and maximum harmony_var = 0.127.

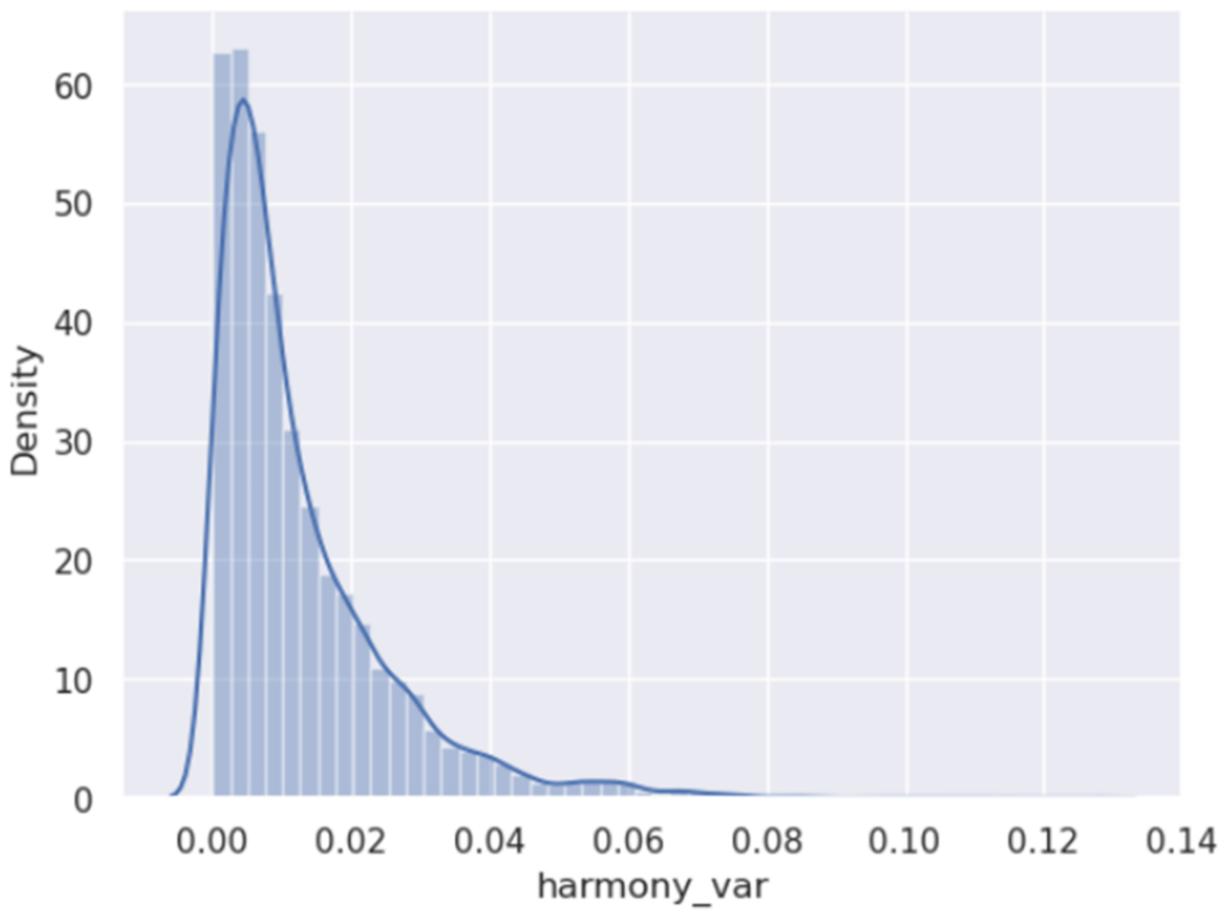
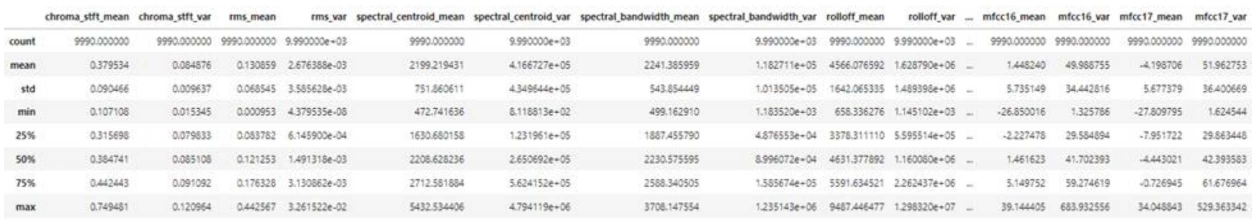


Figure 4-1: Distribution of harmony_var vs Density Graph

4.2 Data Scaling

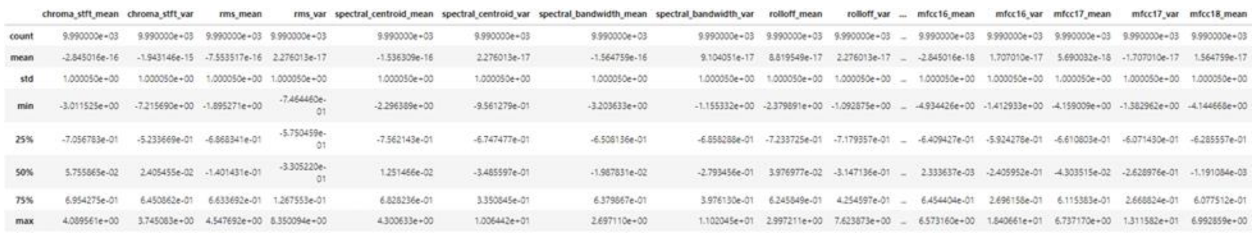
In machine learning, it is often important to scale the input features of a dataset before feeding them to a model. Scaling is a preprocessing technique that ensures that all input features have similar scales and ranges. This can help improve the performance and accuracy of machine learning models by making the optimization process more efficient and effective. The process of scaling involves transforming the input data so that it has a specific distribution or range. One popular scaling method is called standardization, which involves transforming the data so that it has zero mean and unit variance. This means that after scaling, the mean of the data will be zero and the standard deviation will be one. Before scaling, the input features may have different scales and ranges. This can make it difficult for machine learning algorithms to learn the underlying patterns and relationships in the data. For example, a feature that has a large scale or range may dominate the optimization process and overshadow other features that are important for predicting the target variable. To standardize the input data, we can use a built-in method in Python called StandardScaler from the scikit-learn library. This method takes in the input data and scales it using the following formula: z = (x - u) / s where z is the standardized value, x is the original value, u is the mean of the feature, and s is the standard deviation of the feature.

After scaling, the mean of each feature will be zero and the standard deviation will be one. This ensures that all features have a similar scale and range, which can help improve the performance of machine learning models. Hence, standardization is a common scaling technique used in machine learning to transform input features so that they have zero mean and unit variance. The StandardScaler method in scikit-learn is a useful tool for standardizing data and can help improve the performance of machine learning models.



	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_var	...	mfcc16_mean	mfcc16_var	mfcc17_mean	mfcc17_var
count	9990.000000	9990.000000	9990.000000	9.990000e+03	9990.000000	9.990000e+03	9990.000000	9.990000e+03	9990.000000	9.990000e+03	...	9990.000000	9990.000000	9990.000000	9990.000000
mean	0.379534	0.084876	0.130859	2.676388e-03	2199.219431	4.166727e+05	2241.385959	1.182711e+05	4566.076592	1.628790e+06	...	1.448240	49.988755	-4.198706	51.962753
std	0.090466	0.009637	0.068545	3.585628e-03	751.860611	4.349644e+05	543.854449	1.013505e+05	1642.065335	1.489398e+06	...	5.735149	34.442816	5.677379	36.400669
min	0.107108	0.015345	0.000953	4.379535e-08	472.741636	8.118813e+02	499.162910	1.183520e+03	658.336276	1.145102e+03	...	-26.850016	1.325786	-27.809795	1.624544
25%	0.315698	0.079833	0.083782	6.145900e-04	1630.680158	1.231961e+05	1887.455790	4.876553e+04	3378.311110	5.595514e+05	...	-2.227478	29.584894	-7.951722	29.863448
50%	0.384741	0.085108	0.121253	1.491318e-03	2208.628236	2.650692e+05	2230.575595	8.996072e+04	4631.377892	1.160080e+06	...	1.461623	41.702393	-4.443021	42.393583
75%	0.442443	0.091092	0.176328	3.130862e-03	2712.581884	5.624152e+05	2588.340505	1.585674e+05	5591.634521	2.262437e+06	...	5.149752	59.274619	-0.726945	61.676964
max	0.749481	0.120964	0.442567	3.261522e-02	5432.534406	4.794119e+06	3708.147554	1.235143e+06	9487.446477	1.298320e+07	...	39.144405	683.932556	34.048843	529.363342

Figure 4-2 Before Scaling



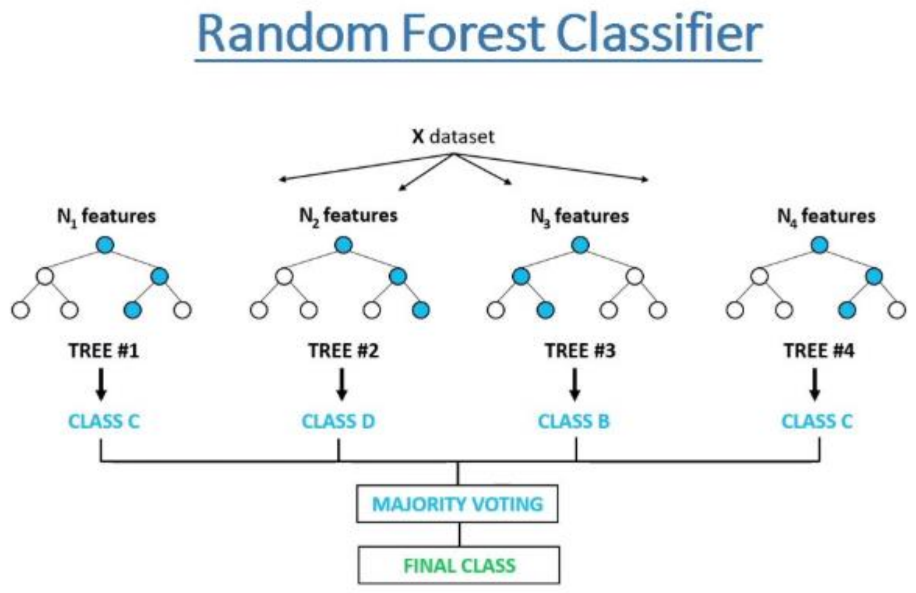
	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var	rolloff_mean	rolloff_var	...	mfcc16_mean	mfcc16_var	mfcc17_mean	mfcc17_var	mfcc18_mean
count	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	...	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03	9.990000e+03
mean	-2.845016e-16	-1.943146e-15	-7.553517e-16	2.276013e-17	-1.536309e-16	2.276013e-17	-1.564759e-16	9.104051e-17	8.819549e-17	2.276013e-17	...	-2.845016e-18	1.707010e-17	5.690032e-18	-1.707010e-17	1.564759e-17
std	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	...	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00	1.000050e+00
min	-3.011525e+00	-7.215690e+00	-1.895271e+00	-7.464460e-01	-2.296389e+00	-9.561279e-01	-3.203633e+00	-1.155332e+00	-2.379891e+00	-1.092875e+00	...	-4.934426e+00	-1.412933e+00	-4.159009e+00	-1.382962e+00	-4.144668e+00
25%	-7.056783e-01	-5.233669e-01	-6.868341e-01	-5.750459e-01	-7.562143e-01	-6.747477e-01	-6.508136e-01	-6.858288e-01	-7.233725e-01	-7.179357e-01	...	-6.409427e-01	-5.924278e-01	-6.610803e-01	-6.071430e-01	-6.285557e-01
50%	5.755865e-02	2.405455e-02	-1.401431e-01	-3.305220e-01	1.251466e-02	-3.485597e-01	-1.987831e-02	-2.793456e-01	3.976977e-02	-3.147136e-01	...	2.333637e-03	-2.405952e-01	-4.303515e-02	-2.628976e-01	-1.191084e-03
75%	6.954275e-01	6.450862e-01	6.633692e-01	1.267553e-01	6.828236e-01	3.350845e-01	6.379867e-01	3.976130e-01	6.245849e-01	4.254597e-01	...	6.454404e-01	2.696158e-01	6.115383e-01	2.668824e-01	6.077512e-01
max	4.089561e+00	3.745083e+00	4.547692e+00	8.350094e+00	4.300633e+00	1.006442e+01	2.697110e+00	1.102045e+01	2.997211e+00	7.623873e+00	...	6.573160e+00	1.840661e+01	6.737170e+00	1.311582e+01	6.992859e+00

Figure 4-3: After Scaling

4.3 Possible Approaches

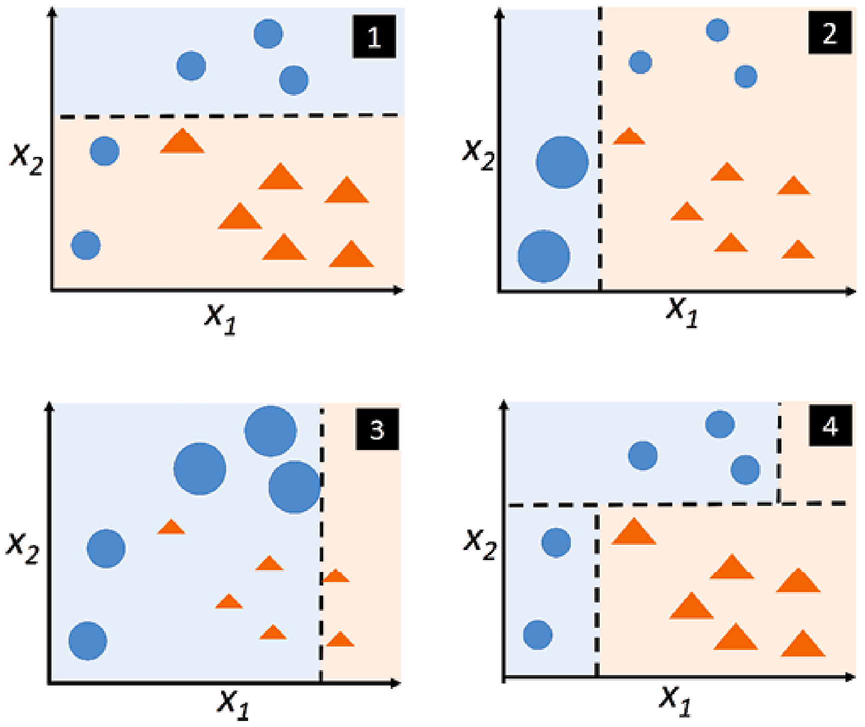
4.3.1 Random Forest

Random forest is a widely used machine learning algorithm for classification and regression tasks. It is an ensemble method that combines multiple decision trees to create a more accurate and robust model. The algorithm is based on decision trees that use a hierarchical structure of nodes to make decisions based on features of the data. Random forest introduces randomness in the tree-building process through random selection of features and random sampling of data points, which helps to reduce overfitting and increase generalization ability. The algorithm is highly accurate, robust, and resistant to outliers and noisy data. Random forest is also easy to use and requires minimal preprocessing of data with few hyperparameters to tune. Overall, random forest is a powerful algorithm that can handle a wide range of machine learning tasks, making it a popular choice in many real-world applications.



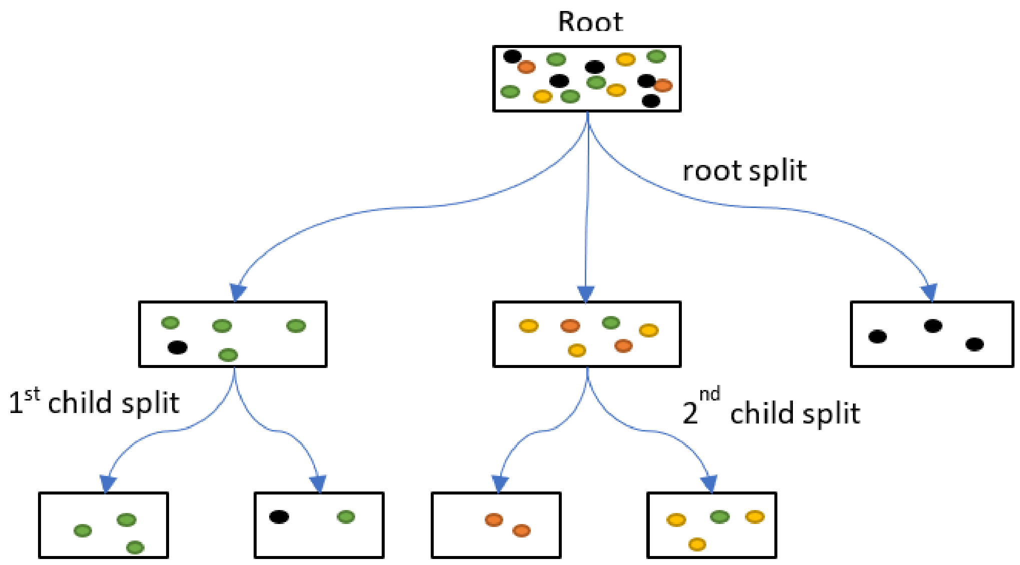
4.3.2 Ada Boost

Adaboost (Adaptive Boosting) is a popular machine learning algorithm that combines multiple weak learners to create a more accurate and robust model. The algorithm is often used for binary classification problems, but can also be extended to multiclass classification and regression tasks. The basic idea behind Adaboost is to iteratively train a series of weak learners on the data, where each subsequent learner gives more weight to the misclassified examples from the previous learner. This adaptive approach allows Adaboost to focus on the difficult examples and improve the model's performance over time. In each iteration, Adaboost assigns a weight to each training example that determines its importance in the next round of training. The weak learner is trained on this weighted dataset, and the model's performance is evaluated. The algorithm then adjusts the weights of the misclassified examples and moves on to the next iteration. The final model is an ensemble of weak learners, weighted according to their performance on the training examples. The weights assigned to each learner depend on their accuracy, with more accurate learners given higher weight.One of the key benefits of Adaboost is its ability to handle complex datasets with high-dimensional features. The algorithm can effectively handle noisy data and outliers, making it a popular choice for many real-world applications. However, Adaboost can be sensitive to overfitting if the weak learners are too complex or the data is too noisy. To avoid this, it is important to tune the hyperparameters and use appropriate regularization techniques.

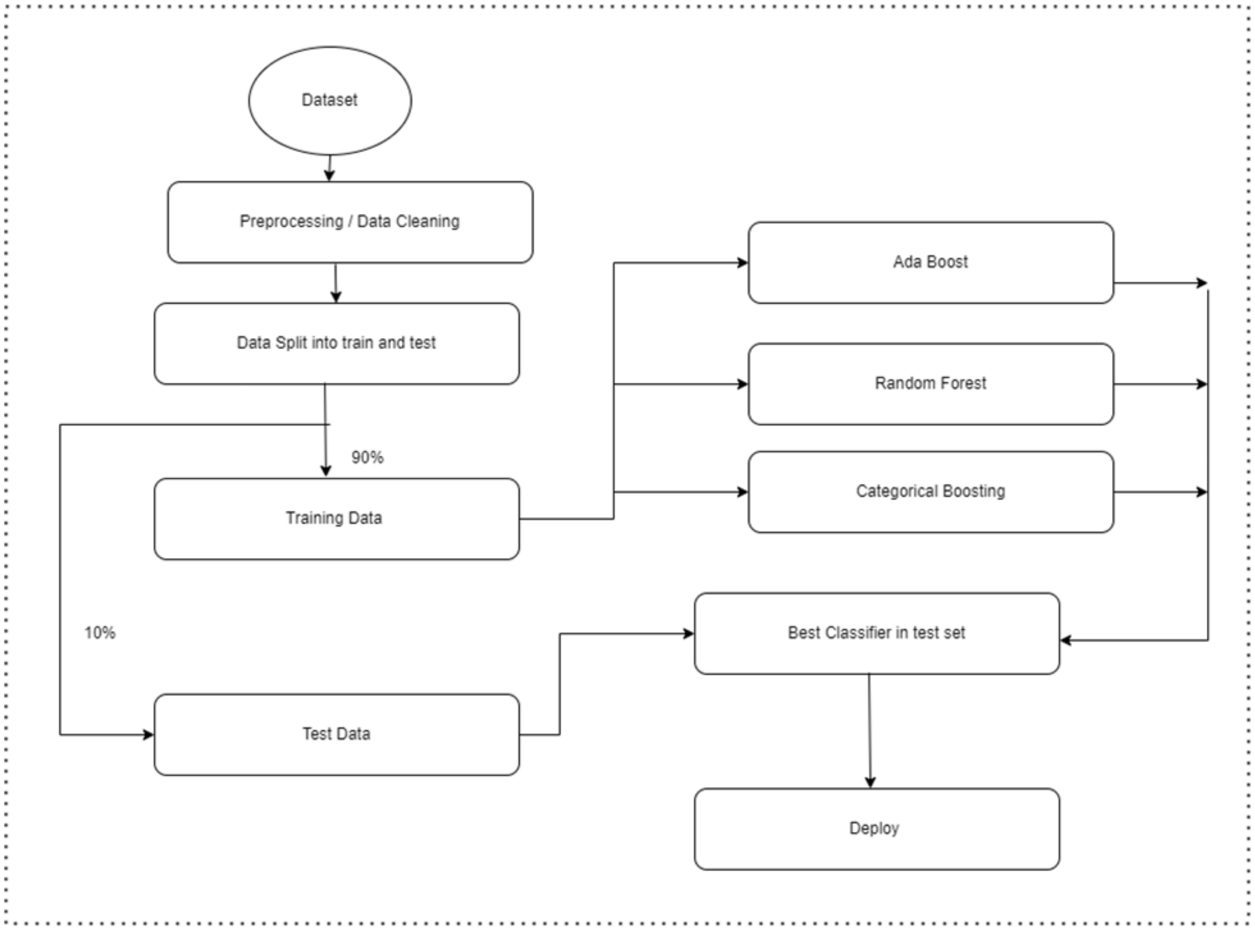


4.3.3 CatBoost

Categorical boosting (CatBoost), is a machine learning algorithm that is designed to handle categorical data and high-dimensional feature spaces. It is a type of gradient boosting algorithm that is based on decision trees, and it was developed by Yandex, a Russian search engine company. The name CatBoost is derived from its ability to handle categorical variables, which are often referred to as "cat" variables.One of the key features of CatBoost is its ability to handle categorical data without the need for one-hot encoding or other preprocessing steps. This is achieved through a combination of methods, including a novel algorithm for gradient computation that is specifically designed for categorical variables.CatBoost also incorporates several other techniques to improve performance and reduce overfitting, including feature scaling, early stopping, and a specialized form of regularization called "ordered boosting". The algorithm also includes several hyperparameters that can be tuned to improve performance, including the learning rate, the depth of the trees, and the number of trees in the ensemble.In addition to its performance on categorical data, CatBoost has been shown to be highly accurate and robust on a wide range of machine learning tasks. It has achieved state-of-the-art results on several benchmark datasets where it outperformed all other methods on a binary classification task. Hence, CatBoost is a powerful and versatile machine learning algorithm that is well-suited to handling high-dimensional data and asdfcategorical variables.



4.4 Block Diagram


Dataset
Preprocessing / Data Cleaning
Data Split into train and test
90%
10%
Training Data
Test Data
Ada Boost
Random Forest
Categorical Boosting
Best Classifier in test set
Deploy

The methodology consists of several steps. Firstly, Data preprocessing is an essential step in machine learning that involves cleaning, transforming, and preparing raw data for analysis. The goal of data preprocessing is to ensure that the data is accurate, complete, and relevant to the problem at hand. The following are the steps involved in data preprocessing:

1) Data Cleaning: This step involves removing missing values, duplicates, and outliers from the dataset.

2) Data Transformation: This step involves converting categorical variables into numerical variables, scaling the data, and normalizing the data.

3) Data Integration: This step involves combining data from multiple sources into a single dataset.

4) Data Reduction: This step involves reducing the number of variables in the dataset by using techniques such as principal component analysis (PCA).

Then, Splitting data into train and test set is done to evaluate the performance of the model on data it has not seen before, which is crucial for ensuring that the model can generalize well to new, unseen data. The process of splitting the data into the training set and testing set is straightforward. Here are the steps to follow:

1) Prepare the dataset

2) Define the size of the training and testing sets

Randomly divide the data: Once you have decided on the size of the training and testing sets, you can randomly divide the data into the two sets. It is crucial to ensure that the split is random to avoid any bias in the dataset. It is crucial to note that the testing set should not be used for training or tuning the model. Doing so will lead to overfitting, where the model performs well on the testing set but poorly on new, unseen data. To avoid overfitting, you can use techniques like cross-validation or hold-out validation. Once the data is split, the next step is to train the classifier. In this case, we will be using three different classifiers: cat boost classifier, voting classifier, and adaboost classifier. Each of these classifiers has its strengths and weaknesses, and we will choose the one with the best performance on the training set. The cat boost classifier is a gradient boosting algorithm that is known for its speed and accuracy. The voting classifier is an ensemble method that combines multiple classifiers to improve performance. The adaboost classifier is a boosting algorithm that focuses on the misclassified samples and tries to improve the performance of the classifier. After training the classifiers, we will evaluate their performance on the testing set. The performance metrics we will use to evaluate the classifiers are accuracy, precision, recall, and F1 score. We will choose the classifier with the best performance on the testing set.

Finally, we will deploy the chosen classifier in a production environment. This involves creating an API that can receive input data and return predictions. The API will be tested thoroughly to ensure that it is working as expected and can handle large volumes of data.

5. Results and Discussion

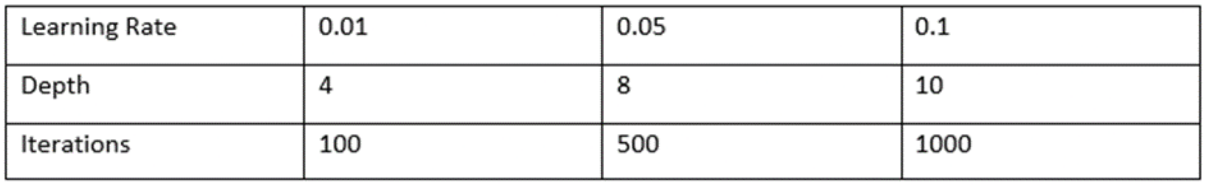
5.1 Experimental Tracking / Model Comparison

Table 1: Various results of different algorithms

Algorithm	Accuracy	Precision	Recall	F1 Score
Cat Boost	0.90590	0.90590	0.90585	0.90579
Random Forest	0.88488	0.886660	0.88484	0.88454
Ada Boost	0.893893	0.895662	0.89383	0.89336
Stochastic Gradient Descent	0.694694	0.702387	0.69457	0.69382
Gaussian Naïve Bayes	0.52052	0.523862	0.52047	0.501154

5.1.1 Hyperparameter Tuning For Cat Boost :

CatBoost is a gradient boosting framework that is designed to handle categorical variables in the data. It is an open-source machine learning library that provides an efficient and scalable implementation of gradient boosting on decision trees. Hyperparameter tuning is a critical step in machine learning to find the optimal values for the hyperparameters of an algorithm. Hyperparameters are parameters that are not learned during the training process, but instead are set prior to training. They include values such as learning rate, depth, and iterations. The best learning rate from the hyperparameter grid is 0.1, which is the step size at which the gradient descent algorithm updates the model weights during training. A higher learning rate can cause the algorithm to overshoot the optimal solution, while a lower learning rate may take longer to converge. The best depth from the hyperparameter grid is 10, which represents the number of levels in the decision tree. A higher depth can lead to overfitting, while a lower depth can result in underfitting. The best number of iterations from the hyperparameter grid is 1000, which represents the number of trees in the gradient boosting algorithm. A higher number of iterations can improve the performance of the algorithm, but also increase the risk of overfitting. In addition to these hyperparameters, CatBoost provides several other hyperparameters that can be tuned for better performance, such as regularization parameters, subsampling rates, and feature importance calculation methods. Overall, hyperparameter tuning is a critical step in the machine learning process to achieve optimal performance from an algorithm like CatBoost. It involves searching through a range of hyperparameters to find the optimal values for the specific problem being addressed. Grid search is a popular hyperparameter tuning method that involves exhaustively searching through a predefined set of hyperparameters to find the best combination.



Learning Rate	0.01	0.05	0.1
Depth	4	8	10
Iterations	100	500	1000

5.1.2 Combined model of voting classifier

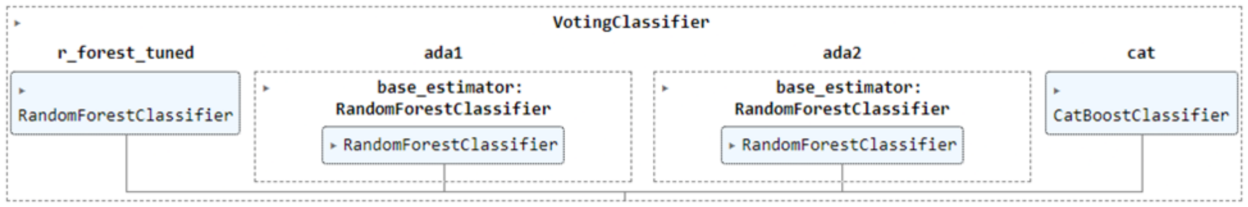
Voting classifiers are a type of ensemble learning method in machine learning, where multiple models are combined to make a final prediction or classification. In a voting classifier, each model makes a prediction or classification on the input data, and the final prediction is determined by a majority vote among the models. This means that the class with the highest number of votes is selected as the final prediction.

There are two types of voting classifiers; namely:

Hard voting: In hard voting, the final prediction is based on the majority vote of the models. For example, if there are three models, and two models predict class A while one model predicts class B, then the final prediction will be class A.

Soft voting: In soft voting, the final prediction is based on the probabilities of the predicted classes. Each model predicts the probability of each class, and the final prediction is based on the average of the predicted probabilities. This can be useful when the models produce probabilistic outputs, such as in logistic regression or neural networks.

Voting classifiers can be useful in improving the accuracy and robustness of machine learning models, especially when individual models have different strengths and weaknesses. They are commonly used in competitions such as Kaggle, and can be implemented in popular machine learning libraries such as scikit-learn.



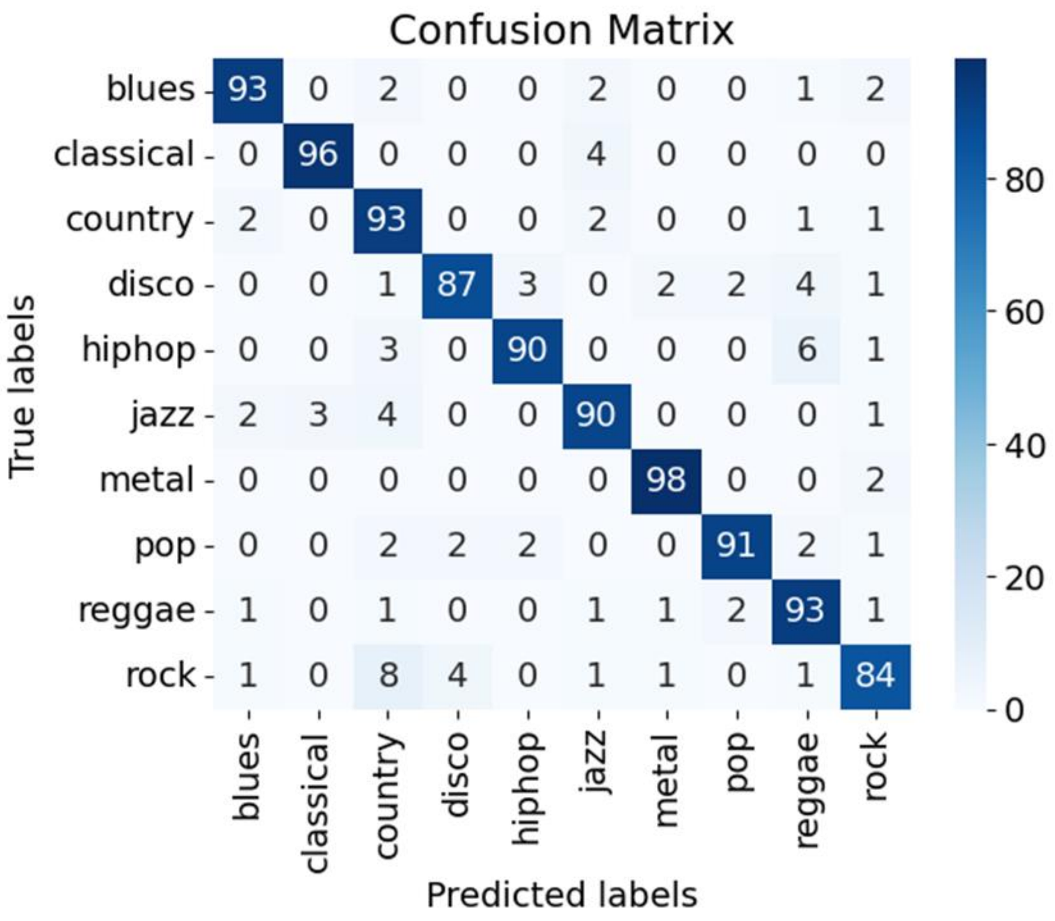
Accuracy, Precision, Recall of Voting Classifier

After conducting an extensive analysis of our model's performance, we have come to the conclusion that the implementation of a voting classifier did not result in any significant improvements. Furthermore, we have found that the voting classifier consumed a significant amount of space, which is not ideal for efficient computational processing. Therefore, after careful consideration, we have decided to opt for the CatBoost classifier as our primary model. This decision was based on the fact that the CatBoost classifier produced comparable results to the voting classifier while utilizing fewer resources. By selecting the CatBoost classifier, we can optimize the computational efficiency of our model without compromising its performance. We believe that this decision will lead to better outcomes for our project and contribute to the continued success of our team.

Table 3: Voting Classifier Metrics

Algorithm	Accuracy	Precision	Recall	F1 Score
Voting Classifier Final	0.90990990990	0.90993517948	0.90987878787	0.90970108680

Confusion matrix for CAT BOOST



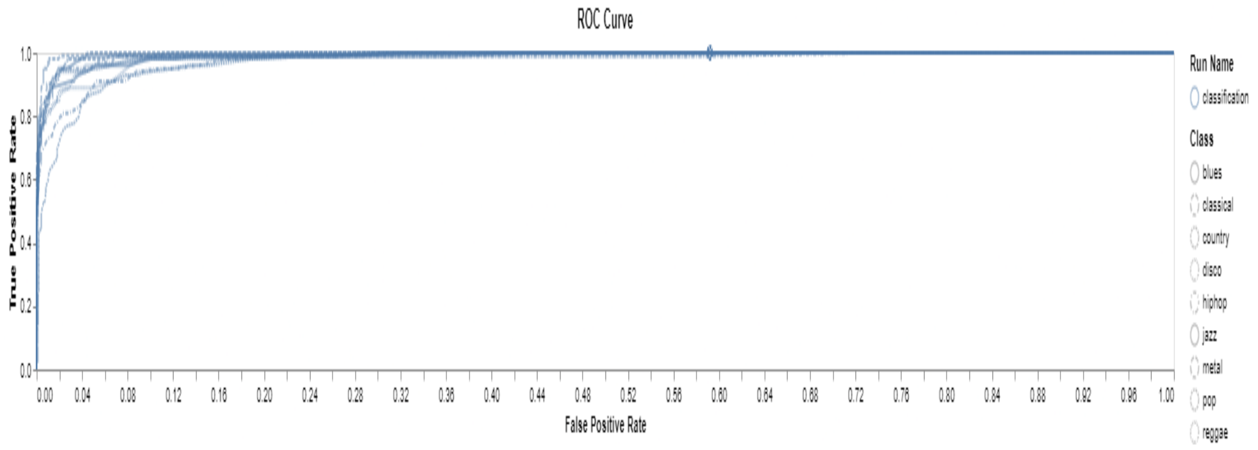
The importance of features can provide insight into the underlying relationships between the input features and the target variable. It can also help identify which features have the most predictive power and can be used to improve the model's performance.In addition, feature importance can be used for feature selection and feature engineering. By identifying the most important features, we can remove irrelevant or redundant features from the dataset and simplify the model. This can lead to improved model performance, reduced computation time, and better interpretability. Feature importance is a technique used in machine learning to determine the relative importance of different features in predicting the target variable. There are several methods for calculating feature importance, and it can provide insights into the underlying relationships between the input features and the target variable. It can also be used for feature selection and engineering to improve the model's performance. In the above, each row represents the true class, and each column represents the predicted class. The diagonal cells represent the number of correct predictions for each class (TP). The off-diagonal cells represent the number of incorrect predictions, where a prediction was made for a class but the true class was something else. For example, the cell in 1st-row blues and column 1st blues represents the number of instances where the true class was 1st column 1st row but other than is not the true class. Similarly, the diagonal is all the true and final labels of the class.From the confusion matrix, we can calculate several performance metrics for each class, such as precision, recall, and F1 score, which can help us understand how well the model is performing for each class. We can also calculate overall metrics, such as accuracy, which tells us how many of the predictions were correct overall.

EXPERIMENT TRACKING FROM WEIGHTS AND BIAS (For Random Forest Classifier)

Weight and Bias (W&B) is a comprehensive platform that provides machine learning teams with end-to-end management and tracking of their ML lifecycle. It helps to deploy and maintain machine learning models efficiently. W&B offers a broad range of features that allow users to visualize, track, and manage their models' performance with ease. In this response, we will focus on explaining the concept of weights and biases in machine learning and how W&B can help in managing them. In machine learning, weights and biases are essential parameters used to define the model's behavior. The model's learning process aims to optimize these parameters to reduce the difference between the predicted output and the actual output. Weights are the coefficients that the model assigns to each input variable, and biases are the constant values that the model adds to the input variables to adjust the output. The values of these parameters are critical in determining the model's accuracy and generalization capabilities. W&B allows users to monitor and visualize the evolution of these parameters during the training process. It helps to identify trends, compare the performance of different models, and make data-driven decisions to improve the model's accuracy. W&B also provides a mechanism for hyperparameter tuning, which is the process of selecting the best values for these parameters to achieve the optimal model performance. One of the advantages of using W&B with MLflow is the powerful visualization platform that it provides. Users can create customized charts and graphs to visualize the model's performance and the evolution of the weights and biases during the training process. These visualizations enable users to quickly identify patterns, anomalies, and trends, and make informed decisions to optimize their models. Hence we have decided to use W&B instead of ML Flow.

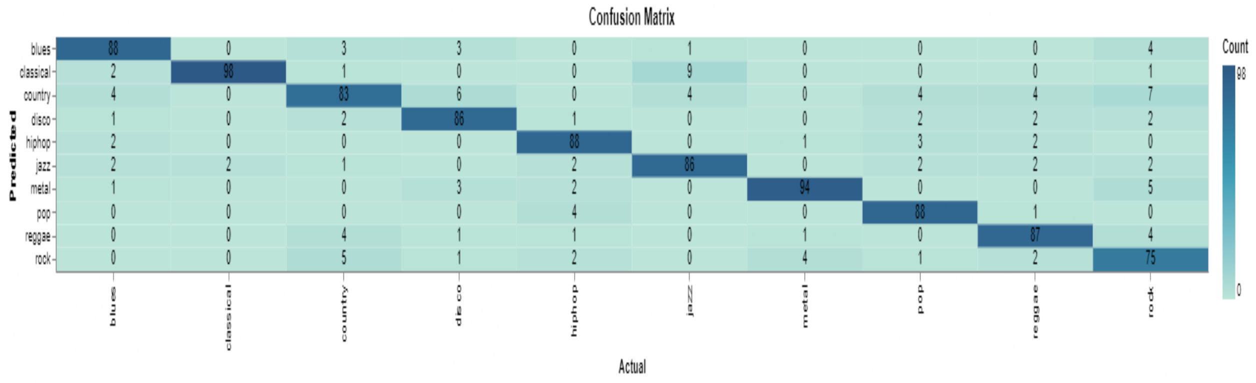
ROC CURVE

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model. It plots the true positive rate (TPR) against the false positive rate (FPR) for different classification thresholds.The TPR, also known as sensitivity or recall, is the proportion of positive cases correctly identified by the model. The FPR is the proportion of negative cases that are incorrectly classified as positive. A perfect classifier would have a TPR of 1 and an FPR of 0, resulting in a point at the top-left corner of the ROC curve. A classifier that performs no better than random guessing would produce a diagonal line from the bottom-left to the top-right corner, with an area under the curve (AUC) of 0.5.



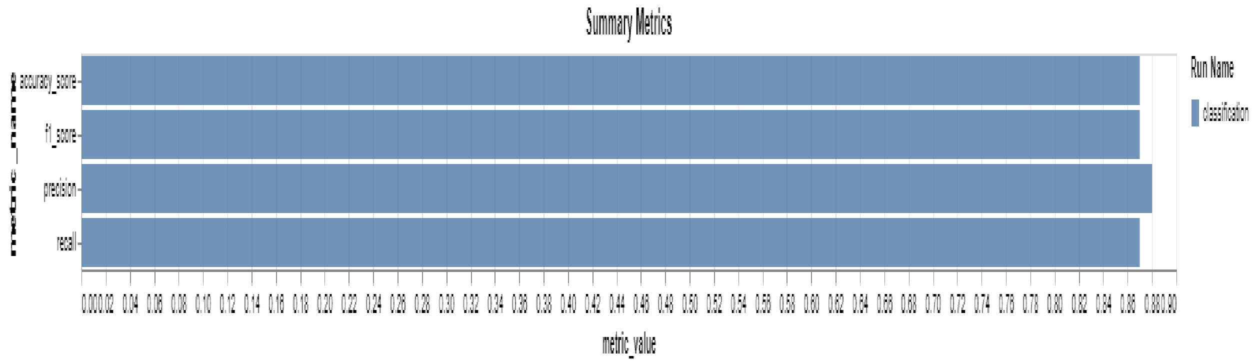
Confusion Matrix

Confusion matrix similar to the above is given by weights and biases hence it is similar to the confusion matrix mentioned above. But finally a confusion matrix offers an evaluation of a classification model's performance in terms of accuracy, precision, recall, and F1-score. These metrics are calculated using the values in the confusion matrix.The confusion matrix is a practical tool that highlights a classification model's strengths and weaknesses. It can help in making informed decisions on how to improve the model's performance.



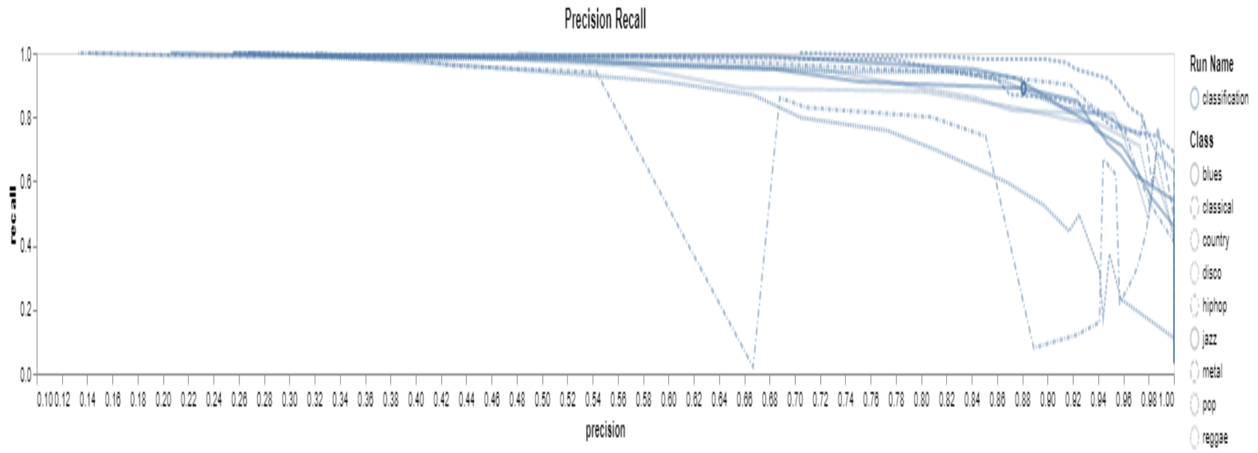
Summary metrics

The summary metrics shows that different metrics values of the random forest algorithm. We have used a simple random forest classifier in this and the accuracy is around 86% of the classifier. Similarly, the f1 score ranges to the same value of around 86% . The highest value is the precision which means that high precision means that the value or result obtained is very close to the true or expected value, with a low level of error or uncertainty. Similarly the recall value is around 86% of the simple random forest classifier.



Precision-Recall Curve

The Precision-Recall (PR) curve is a popular graphical evaluation method for machine learning models, particularly for binary classification problems. The PR curve is a plot of precision versus recall values for a range of classification thresholds. Precision and recall are two performance metrics that are commonly used to evaluate binary classification models. Precision measures the proportion of true positive predictions among all positive predictions made by the model. In other words, it measures the accuracy of the positive predictions. Recall, on the other hand, measures the proportion of true positives predicted by the model among all actual positive samples in the dataset. It is a measure of the completeness of the positive predictions. To plot a PR curve, we first generate a set of classification scores or probabilities for the positive class (class 1) for all samples in the test set. We then rank the samples based on their scores in descending order and set a range of classification thresholds. For each threshold value, we assign a label of 1 or 0 to each sample based on whether its score is above or below the threshold. For each threshold, we calculate the precision and recall values based on the predicted labels and the true labels of the samples in the test set. We then plot the precision-recall pairs for all the thresholds, which generates a curve that shows how the precision and recall values change as the classification threshold is varied. A perfect classifier would have a PR curve that is a square with the top right corner representing perfect precision and recall. However, most classifiers are not perfect and will have a curve that is somewhere between the diagonal line representing random guessing and the perfect square. The curve will typically slope downward from left to right, with higher precision values corresponding to lower recall values.The PR curve can be used to evaluate the performance of a binary classification model, and to compare the performance of different models. The area under the PR curve (AUC-PR) is a commonly used summary statistic that measures the overall performance of the model. A higher AUC-PR indicates better performance, with a maximum value of 1 for a perfect classifier.In summary, the precision-recall curve is a useful graphical tool for evaluating binary classification models. It provides insight into the trade-off between precision and recall, and can help identify the optimal classification threshold for a given problem.



Learning Curve

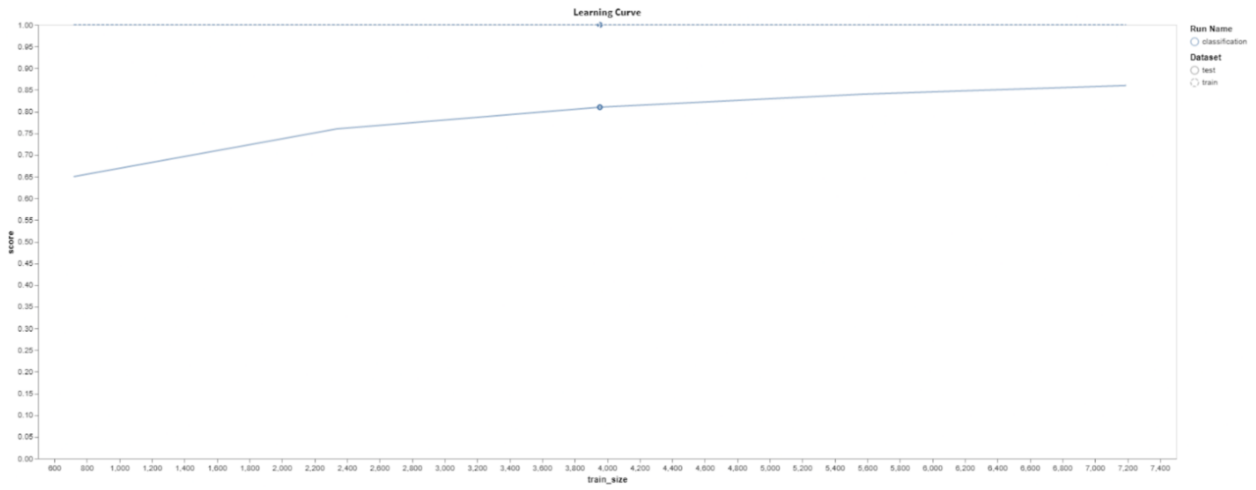
A learning curve is a graphical representation of the relationship between the size of the training set and the performance of a machine learning model. The x-axis of the curve represents the number of samples used for training, while the y-axis represents the model's performance score on a specific metric, such as accuracy or mean squared error. Learning curves are typically used to evaluate how well a model can generalize to new data as more training data is added. They are also useful for identifying whether a model is overfitting or underfitting the training data.To create a learning curve, we start by splitting the available data into a training set and a validation set. We then train the model on increasingly larger subsets of the training set and evaluate its performance on the validation set at each iteration. The performance score can be calculated using any appropriate metric for the specific problem, such as accuracy, precision, recall, or F1 score.As the training set size increases, the model's performance score on the validation set typically improves initially, as it is able to learn more patterns and generalize better. However, at a certain point, the model's performance may plateau, and adding more data may not result in any significant improvement. This is known as the model's convergence point.The learning curve can help identify several important aspects of the model's performance:

Bias or underfitting

Variance or overfitting

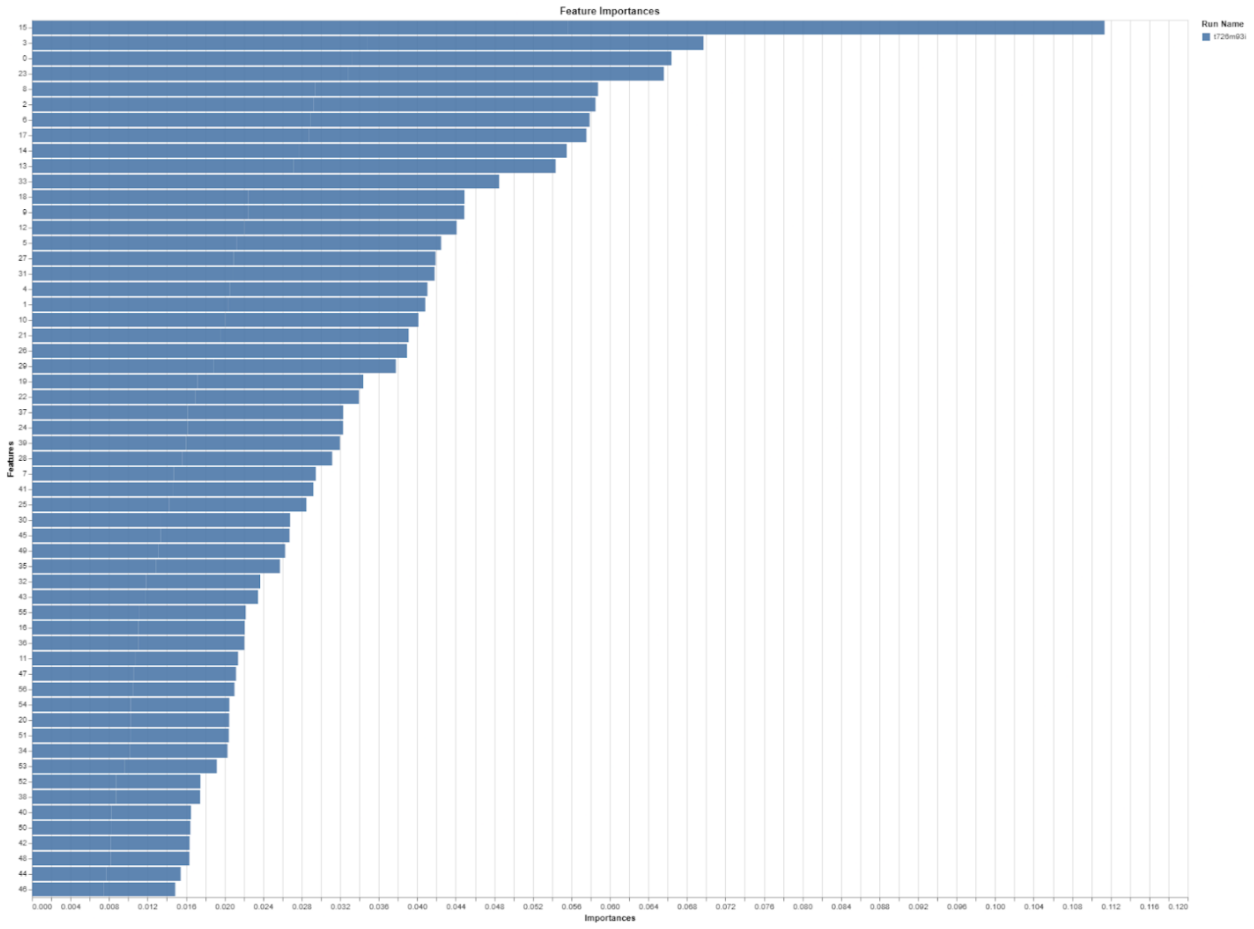
Optimal training set size

Hence, learning curves are a useful tool for evaluating the performance of machine learning models and identifying whether they are underfitting or overfitting the data. They can also help determine the optimal training set size required for the model to achieve good performance on new data.



Feature Importance

Feature importance refers to a technique used in machine learning to determine the relative importance of different features or variables in predicting the target variable or outcome. The importance of each feature is evaluated based on its impact on the model's predictive performance.Feature importance can be calculated using a variety of methods, depending on the algorithm used for modeling. The importance of features can provide insight into the underlying relationships between the input features and the target variable. It can also help identify which features have the most predictive power and can be used to improve the model's performance.In addition, feature importance can be used for feature selection and feature engineering. By identifying the most important features, we can remove irrelevant or redundant features from the dataset and simplify the model. This can lead to improved model performance, reduced computation time, and better interpretability. Therefore, feature importance is a technique used in machine learning to determine the relative importance of different features in predicting the target variable. There are several methods for calculating feature importance, and it can provide insights into the underlying relationships between the input features and the target variable. It can also be used for feature selection and engineering to improve the model's performance.



6. Conclusion

Music genre classification project presented in this report demonstrated the successful application of machine learning models and techniques to classify audio clips into different music genres. The project showcased the importance of data collection and preprocessing, including the selection of appropriate datasets, cleaning, normalization, and feature engineering, in achieving accurate classification results. The project also emphasized the importance of selecting appropriate performance metrics, interpreting and visualizing the results, tracking experiments, and benchmarking the model's performance. The project's results showed promising accuracy in classifying audio clips into different music genres, indicating that machine learning models can be effective tools in solving music genre classification problems. However, the project also highlighted some challenges, such as subjective labeling, limited domain expertise, bias in labeling, and limited scalability, which need to be addressed to achieve better performance. Overall, this project contributes to the field of music genre classification and has the potential to be extended to other applications, such as music recommendation systems. The success of this project opens up opportunities for future research in the area of machine learning and music analysis.