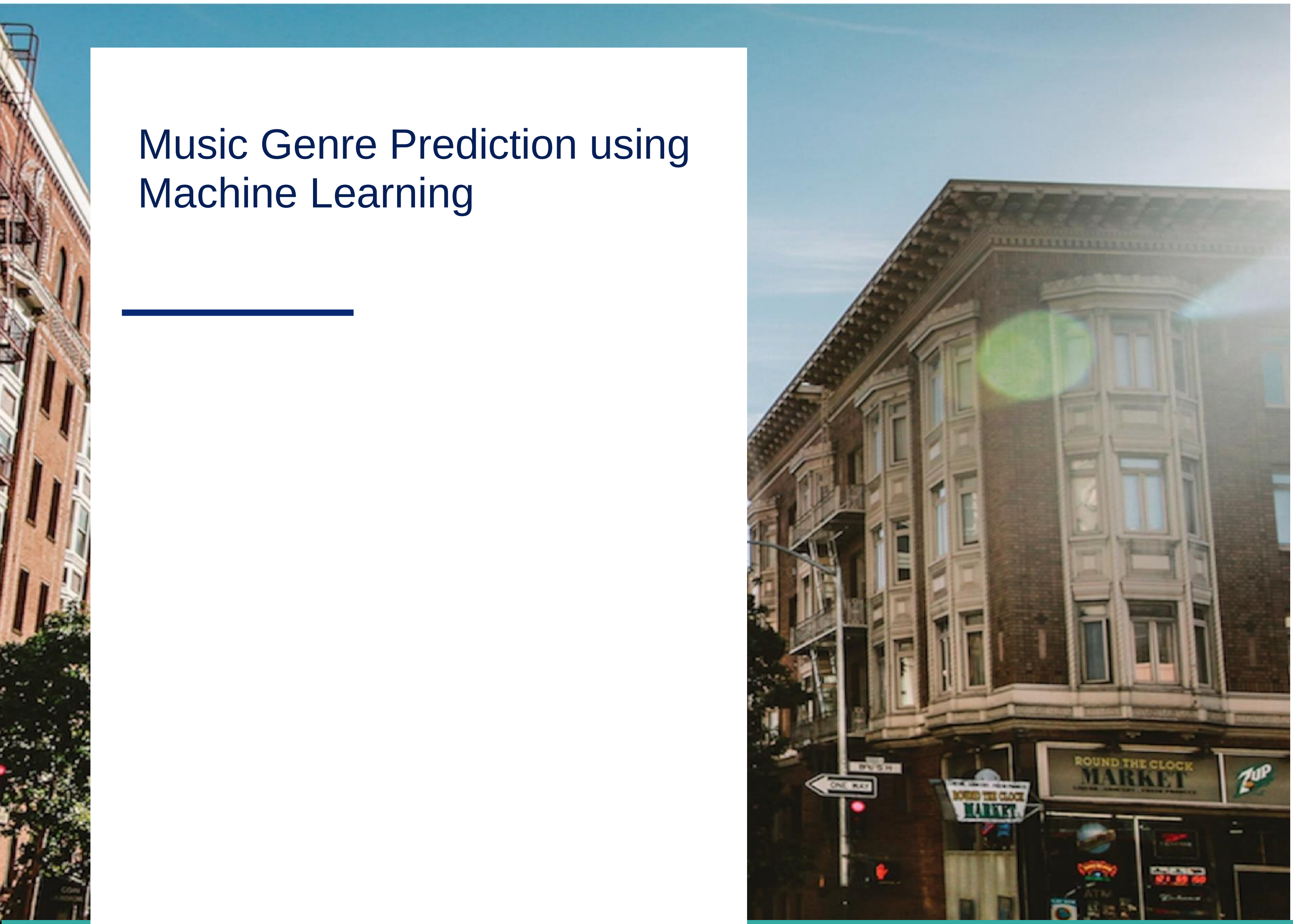


# Music Genre Prediction using Machine Learning

**PSN: 40022593**

LTTS

Authored by: **Adhokshaj Wategaonkar**

## *Problem Statement*

### Music Genre Prediction using Machine Learning

Using the dataset and features of music tracks, predict the genre of music.
Explain and elaborate on the data processing and create a function to take various feature values as inputs and display in output the music genre to the user.
Experiment on various algorithms to get the optimum accuracy & model.
The algorithms selected are ***Support Vector Classifier, Decision Tree Classifier, K-Nearest Neighbors & Random Forest***. Experiment with new algorithms if required.

## *Data Set*

https://www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre

The dataset contains set of audio features provided by ***Spotify***.
Dataset consists of 18 distinct columns and 506 rows.
The distinct columns are instance_id, artist_name, track_name, popularity, acousticness, danceability, duration_ms, energy, instrumentalness, key, liveness, loudness, mode, speechiness, tempo, obtained_date, valence, music_genre.

The full list of genres we want to predict included in the CSV are 'Electronic', 'Anime', 'Jazz', 'Alternative', 'Country', 'Rap', 'Blues', 'Rock', 'Classical', 'Hip-Hop'.

The detailed description of data is:

```
Int64Index: 50000 entries, 0 to 50004
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   instance_id       50000 non-null  float64
 1   artist_name       50000 non-null  object 
 2   track_name        50000 non-null  object 
 3   popularity        50000 non-null  float64
 4   acousticness      50000 non-null  float64
 5   danceability      50000 non-null  float64
 6   duration_ms       50000 non-null  float64
 7   energy            50000 non-null  float64
 8   instrumentalness  50000 non-null  float64
 9   key               50000 non-null  object 
 10  liveness          50000 non-null  float64
 11  loudness          50000 non-null  float64
 12  mode              50000 non-null  object 
 13  speechiness       50000 non-null  float64
 14  tempo             50000 non-null  object 
 15  obtained_date     50000 non-null  object 
 16  valence           50000 non-null  float64
 17  music_genre       50000 non-null  object 
dtypes: float64(11), object(7)
memory usage: 7.2+ MB
```

## *Model and Algorithms*

The problem statement and dataset are the parameters on which we decide what algorithms to use or if it’s going to be a supervised learning or unsupervised learning instance.

Thus, after analyzing the data, we have reached our conclusions and assessments of the dataset.
Clearly here, as we want to predict the music genre based on the features provided which include the characteristics of various music tracks. We have the inputs as well as outputs of the dataset. We can easily divide the entirety of the dataset in training and testing sets. Thus, it is an instance of supervised learning model.

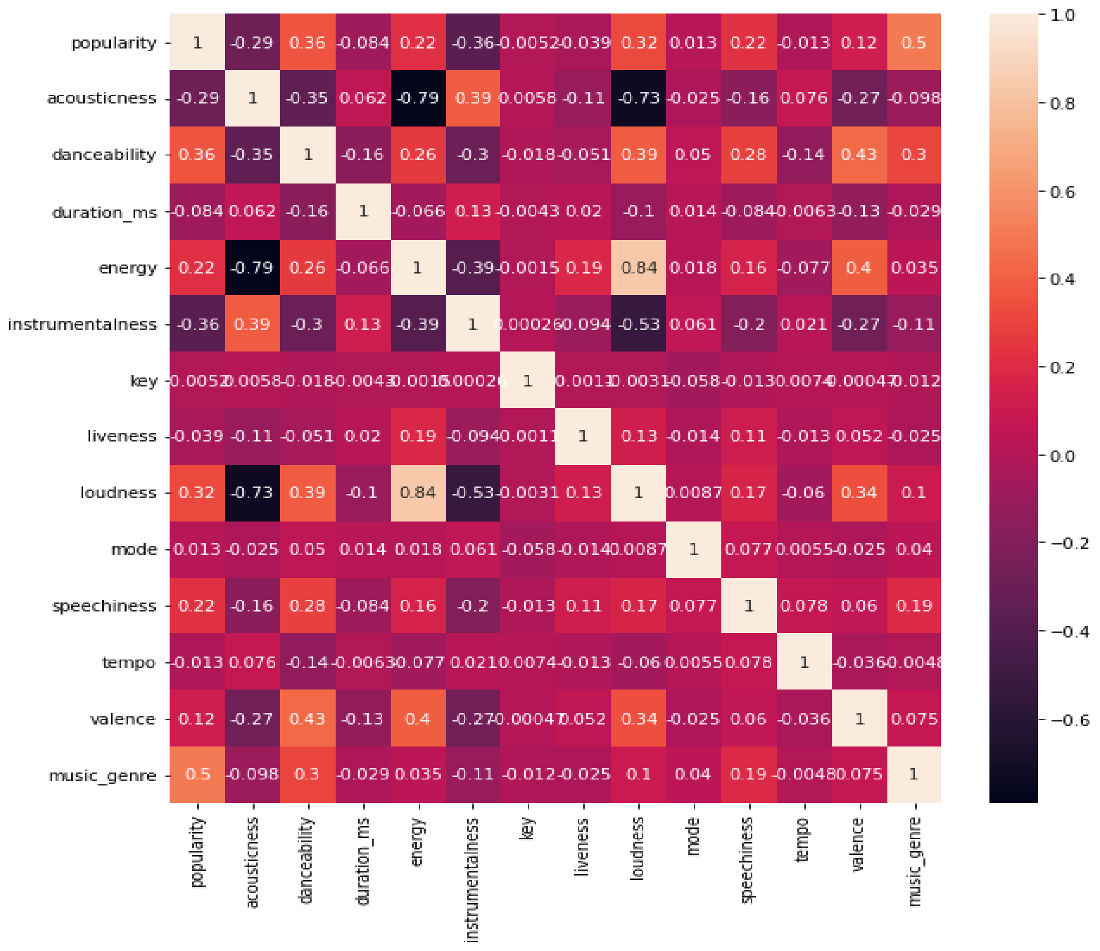
To start, some of the most popular learning algorithms used for supervised learning are Random Forest Classifier, K Neighbors Classifier, Decision Tree Classifier & Support Vector Classifier.

## *Experiments and Implementations*

To begin with let's analyze the features and algorithms we are trying to predict based on all parameters.

The entire feature list is: 'popularity', 'acousticness', 'danceability', 'duration_ms', 'energy', 'instrumentalness', 'liveness', 'loudness', 'speechiness', 'valence', 'key', 'mode', 'tempo'.

As we know, most of these features don't affect the results of our prediction and need to be dropped. So, for feature selection we can start with understanding which of these features have the most effect on music genre. We have used matplotlib to plot a correlation heat map for all the features.



As we can see in the correlation plot, none of the features have an extreme correlation with ‘music_genre’ which we are trying to predict. Thus, for predicting music genre we can safely assume that feature selection can be a bit of hassle. Some experimentation has shown us that we aren't getting higher levels of accuracy using combination of various features and algorithms. The experiments go as follows:

## *Experiments*

1. First Experiment is with using features which correlate to music genre the most i.e., instrumentalness, acousticness, tempo, key & liveness.

   - The accuracy scores we received are:
     1. Accuracy of Random Forest Classifier: **32.24**
     2. Accuracy of KNN Classifier: **16.43**
     3. Accuracy of DecisionTreeClassifier: **31.20**
     4. Accuracy of Support Vector Classifier: **14.48**

2. As we have already seen that using features which have the most correlation is giving us a substandard accuracy score. Thus, in second experiment we will use all the features.

   - The accuracy scores we received are:
     1. Accuracy of Random Forest Classifier: **52.07**
     2. Accuracy of KNN Classifier: **16.34**
     3. Accuracy of DecisionTreeClassifier: **49.21**
     4. Accuracy of Support Vector Classifier: **16.06**

As we can see not much is changing.
**P.S. this is an example. I have experimented in 3-4 iterations by applying different combinations of features and not much changed*.

3. Now to try something new we are changing the random state (RS) and the test data split (TDS) to get different results.

   - The accuracy scores received with RS = 43 and TDS = 0.3 are:
     1. Accuracy of Random Forest Classifier: **50.87**
     2. Accuracy of KNN Classifier: **16.70**
     3. Accuracy of DecisionTreeClassifier: **48.38**
     4. Accuracy of Support Vector Classifier: **15.93**

   - The accuracy scores received with RS = 99 and TDS = 0.15 are:
     1. Accuracy of Random Forest Classifier: **50.22**
     2. Accuracy of KNN Classifier: **16.81**
     3. Accuracy of DecisionTreeClassifier: **47.73**
     4. Accuracy of Support Vector Classifier: **16.41**

We can see the accuracy reducing even further after this experiment.
Thus, trying new algorithms which aren't present in problem statement we selected.

The new algorithms we used are: Logistic Regression (LR), Extreme Gradient Boosting (XGBoost), Light Gradient boosting (LGBoost).

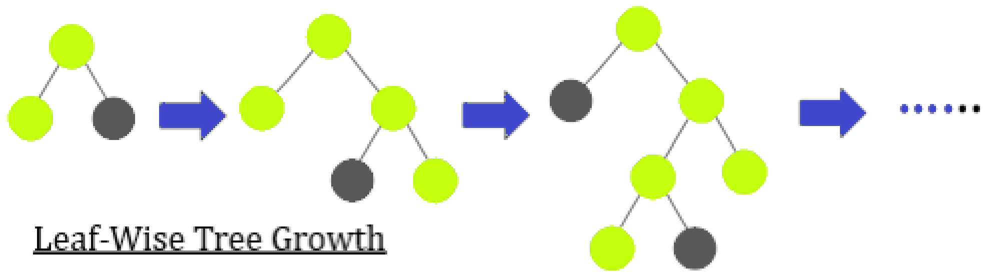
   - The accuracy scores we received with RS = 99 and TDS = 0.15 are:
     1. Accuracy of Logistic Regression: **24.78**
     2. Accuracy of Extreme Gradient Boosting Machine: **55.77**
     3. Accuracy of Light Gradient Boosting Machine: **57.54**

Thus, in accordance with all the data we have, we can assume that Light Gradient Boost Machine (LGBoost) is the most accurate algorithm we can implement in the designated time for experimentation.

## *Light Gradient Boosting (LGBM):*

LightGBM is a gradient boosting framework based on decision trees to increases the efficiency of the model and reduces memory usage.

LGBM splits the tree leaf-wise as opposed to other boosting algorithms that grow tree level-wise. It chooses the leaf with maximum delta loss to grow. Since the leaf is fixed, the leaf-wise algorithm has lower loss compared to the level-wise algorithm. Leaf-wise tree growth might increase the complexity of the model and may lead to overfitting in small datasets. As we have a larger dataset, we can use LGBM optimally.



### Parameter Tuning:

Few important parameters and their usage are listed below:

- max_depth: It sets a limit on the depth of tree. The default value is 20. It is effective in controlling over fitting.
- categorical_feature: It specifies the categorical feature used for training model.
- bagging_fraction: It specifies the fraction of data to be considered for each iteration.
- num_iterations: It specifies the number of iterations to be performed. The default value is 100.
- num_leaves: It specifies the number of leaves in a tree. It should be smaller than the square of max_depth.
- max_bin: It specifies the maximum number of bins to bucket the feature values.
- min_data_in_bin: It specifies minimum amount of data in one bin.
- task: It specifies the task we wish to perform which is either train or prediction. The default entry is train. Another possible value for this parameter is prediction.
- feature_fraction: It specifies the fraction of features to be considered in each iteration. The default value is one.

## *Ensemble Learning:*

Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model. The combined models increase the accuracy of the results significantly. There are three main methods of ensemble learning: bagging, stacking & boosting.

### **Boosting:**

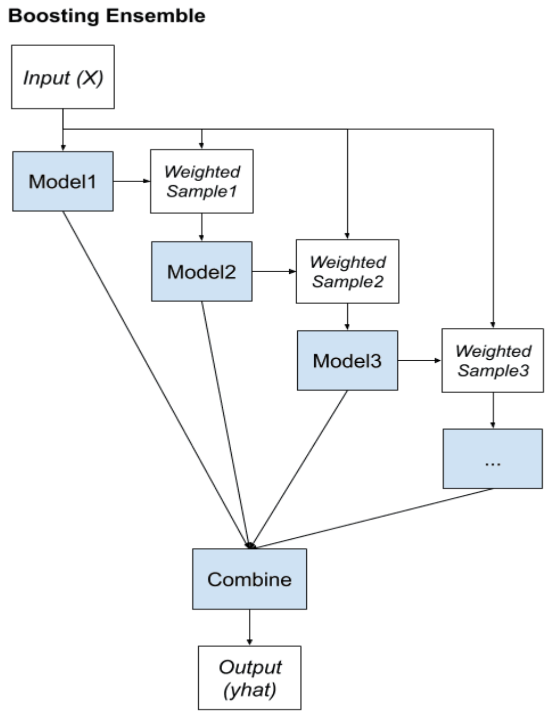
In boosting, the training dataset for each subsequent classifier increasingly focuses on instances misclassified by previously generated classifiers. The key property of boosting ensembles is the idea of correcting prediction errors. The models are fit and added to the ensemble sequentially such that the second model attempts to correct the predictions of the first model, the third corrects the second model, and so on.

The technique combines several weak base learners to form one strong learner, thus significantly improving the predictability of models. Boosting works by arranging weak learners in a sequence, such that weak learners learn from the next learner in the sequence to create better predictive models.

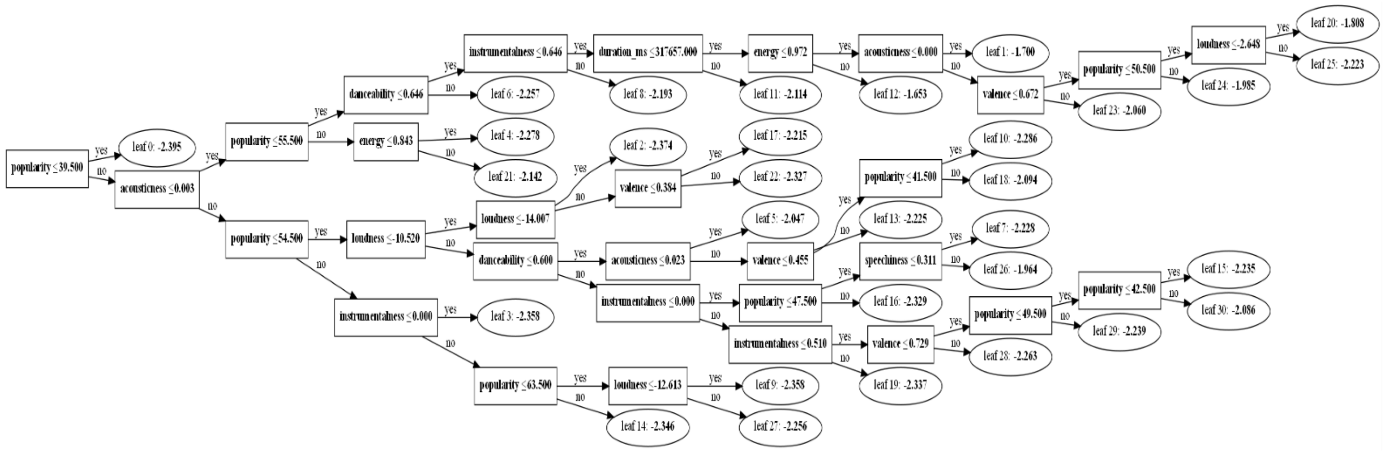
This typically involves the use of very simple decision trees that only make a single or a few decisions, referred to in boosting as weak learners. The predictions of the weak learners are combined using simple voting or averaging, although the contributions are weighed proportional to their performance or capability. The objective is to develop a so-called "*strong-learner*" from many purpose-built "*weak-learners*."

LGBoost (Light Gradient Boosting) uses weak learners in the form of decision trees, which mostly include one split that is popularly known as decision stumps.

Here is a demonstration of how Boosting works:



For visualizing how LGBM is working on our model we have plotted a decision tree:



Go figure :)

*P.S. refer code for image clarity*

## *Hyperparameter tuning in LGBM*

Generally, hyperparameters of the most tree-based models can be grouped into 4 categories:

1. Parameters that affect the structure and learning of the decision trees
2. Parameters that affect the training speed
3. Parameters for better accuracy
4. Parameters to combat overfitting

Complete model optimization includes many different operations:

1. Choosing the optimal starting hyperparameters for your algorithm (conditional on the task type and data stats)
2. Defining the hyperparameters to optimize, their grid and distribution
3. Selecting the optimal loss function for optimization
4. Configuring the validation strategy
5. Further optimization (e.g., n_estimators tuning with early_stopping for tree ensembles like LGBM)
6. Results analysis

Most of the time, these categories have a lot of overlap, and increasing efficiency in one may risk a decrease in another. That's why tuning them manually is a giant mistake and should be avoided.

Thus, we figured out a shortcut :)

## *Optuna*

Optuna is a state-of-the-art automatic hyperparameter tuning framework that is completely written in Python.

## *What is Optuna?*

Optuna uses something called define-by-run API which helps the user to write high modular code and dynamically construct the search spaces for the hyperparameters. Different samplers like grid search, Random, Bayesian and Evolutionary algorithms are used to automatically find the optimal parameter.

- **Grid Search**: It searches the predetermined subset of the whole hyperparameter space of the target algorithm.
- **Bayesian**: This method uses a probability distribution to select a value for each of the hyperparameters.
- **Random Search:** As the name suggests randomly samples the search space until the stopping criteria are met.
- **Evolutionary Algorithms**: The fitness function is used to find the values of the hyperparameters.

Using Optuna on our code we got the following results:

```
learning_rate                     : 0.03
num_leaves                        : 101
colsample_bytree                  : 0.9499233962132211
subsample                         : 0.785506374579681
verbosity                         : -1
random_state                      : 42
objective                         : regression
metric                            : l2
num_threads                       : 6
reg_alpha                         : 0.011962330995271336
min_sum_hessian_in_leaf           : 0.0019187726860649189
reg_lambda                        : 8.4596479674439e-08
n_estimators                      : 143
```

Thus, if we set our LGBM model to these Hyperparameters it will give us optimum results.

## *Results*

1. Random Forest gives the highest accuracy among selected algorithms. But it is still less than what one should expect.

2. SVM takes longer time to train than any other algorithm. SVMs are based around a kernel function. Most implementations explicitly store this as an NxN matrix of distances between the training points to avoid computing entries repeatedly. Thus, making the process time consuming

3. SVM also gives us the least accuracy among all the models. Likely reason being too many training examples to make linear divisions. The limits or divisions stretching too thin.

4. While experimenting with new algorithms, Light Gradient Boosting (LGBM) gives the highest accuracy i.e., 58%.

5. As LGBM is a forward learning ensemble method, LGBM is custom built for handling large datasets without any complexity and it is a perfect fit for our problem statement.

6. After referencing the data available on internet, we can also observe a pattern where accuracy for most models hasn’t surpassed the ballpark of 60%. Thus, we can safely assume that 58% accuracy is decent.

## *Conclusion*

Thus, we can conclude that using the most optimum algorithm (LGBM), we can get an accuracy up to 58% and predict the music genre by inputting values of selected features.

## *Improvement Scope*

When we input feature values outside our training scope, the algorithm still classifies it in a music genre. Can further research on what we can call music and what feature values can be classified as noise.

## References

1. https://www.kaggle.com/datasets/vicsuperman/prediction-of-music-genre
2. www.javatpoint.com
3. www.kaggle.com
4. www.medium.com
5. www.analytics-vidhya.com
6. www.coursera.org
7. www.geeksforgeeks.com
8. www.towardsdatascience.com
9. www.machinelearningmastery.com
10. www.danilzherebtsov.medium.com/effortlessly-tune-lgbm-with-optuna-49de040d0784