Open IIT Data Analytics Team Number: 30

Report

Index

OPEN IIT DATA ANALYTICS REPORT

Abstract

There are multiple factors that affect a song's popularity that can be both related and unrelated to a song's musicality, from the key and modality in which it's written, to the duration of song and year of release. We attempted to build Machine Learning models to predict song's popularity based on available features. Our model included custom and pre-existing features of a song such as its key, modality, loudness, energy and dynamics variation among others. Our model was trained on dataset of around 12 thousand tracks and tested on 4000 samples. Our dataset consisted of 15 features and 1 target variable to be predicted. Predicting how popular a song will be is no easy task because there may be possible to have two songs with same characteristics but one gets famous while other not.

Problem Statement

One firm wants to purchase the rights of a music track by putting bids on them and want to generate maximum revenue out of it. So given are the features of song tracks and we have to predict songs popularity.

Here is the relation between popularity of song, its bid price and expected revenue.

Popularity	Bid price	Expected Revenues
Very high	5	10
High	4	8
Average	3	6
Low	2	4
Very low	1	2

All values in 10,000 $

Platform: Google Colab, Tableau

Language: Python

Libraries and packages: NumPy, Pandas, Matplotlib, Sklearn

Dataset description

We are given 17 columns as features attributes of song tracks.

Column Name	Data type	Range	Description
Id	Integer	1 to 16227	Unique ID of each song
Accousticness	Float	0 to 1	A confidence measure whether the track is acoustic.
Danceability	Float	0 to 1	Describes how suitable a track is for dancing
Energy	Float	0 to 1	Represents perceptual measure of intensity and activity.
Explicit	String	-	
Instrumentalness	Float	0 to 1	Predicts whether a track contains no vocals
Key	Integer	0 to 11	The estimated overall key of the track.
Liveness	Float	0 to 1	Detects the presence of an audience in the recording
Loudness	Float	-43.73 to 1	The overall loudness of a track in decibels (dB)
Mode	String	-	Indicates the modality (major or minor) of a track
Release date	Date	-	Date of release of track
Speechiness	Float	0 to 1	Detects the presence of spoken words in a track
Tempo	Float	0 to 217.91	Pace of a track in beats per minute (BPM)
Valence	Float	0 to 1	Describes the musical positiveness conveyed by a track.
Year	Integer	1920 to 2021	Year of release of track
Duration	Float	0.2 to 80	Duration of the track in minutes
Popularity	String	-	Target variable – how popular song is

Dataset shape

Dataset	Rows	Columns
Train	12227	17
Test	4000	16 (excluding popularity)
Total	16227	17

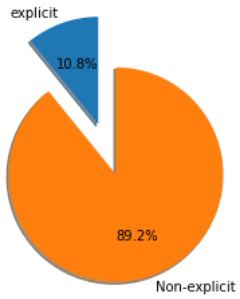
There were no missing values found

Statistics of numerical columns

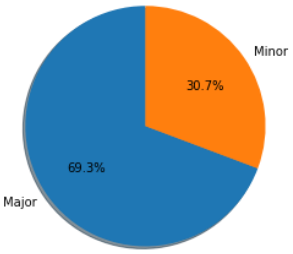
Column name	Minimum	Maximum	Average	Median	Standard Deviation
Accousticness	0.000001	0.996	0.430578	0.354	0.366893
Danceability	0	0.98	0.556353	0.569	0.175373
Energy	0.00002	1	0.522129	0.534	0.262482
Instrumentalness	0	1	0.149321	0.000115	0.297954
Key	0	11	5.205202	5	3.526954
Liveness	0.0147	0.997	0.201365	0.132	0.173987
Loudness	-43.738	1.006	-10.668687	-9.584	5.506888
Speechiness	0	0.968	0.09768	0.0456	0.155895
Tempo	0	216.843	118.167495	116.915	30.200064
Valence	0	1	0.5253	0.532	0.258205
Year	1920	2021	1984.517298	1987	25.911998
Duration(min)	0.2	72.8	3.888133	3.6	2.383133

Exploratory Data Analytics

Distribution of values in columns

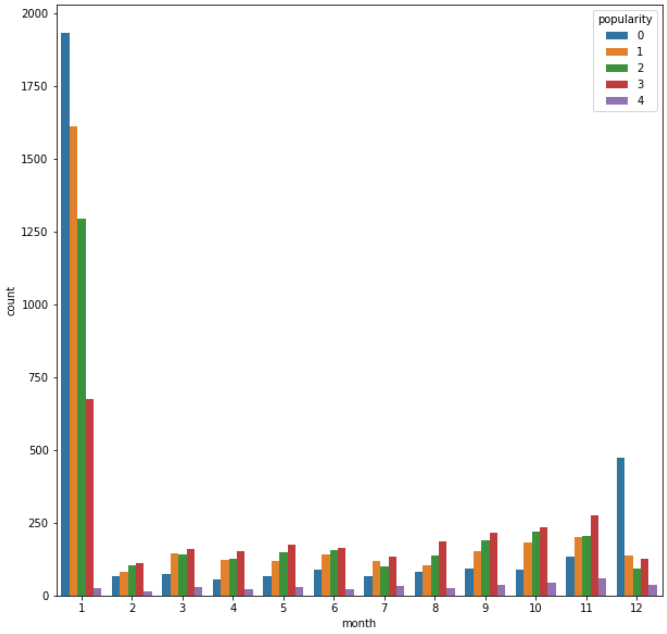


Explicit and Non-explicit tracks



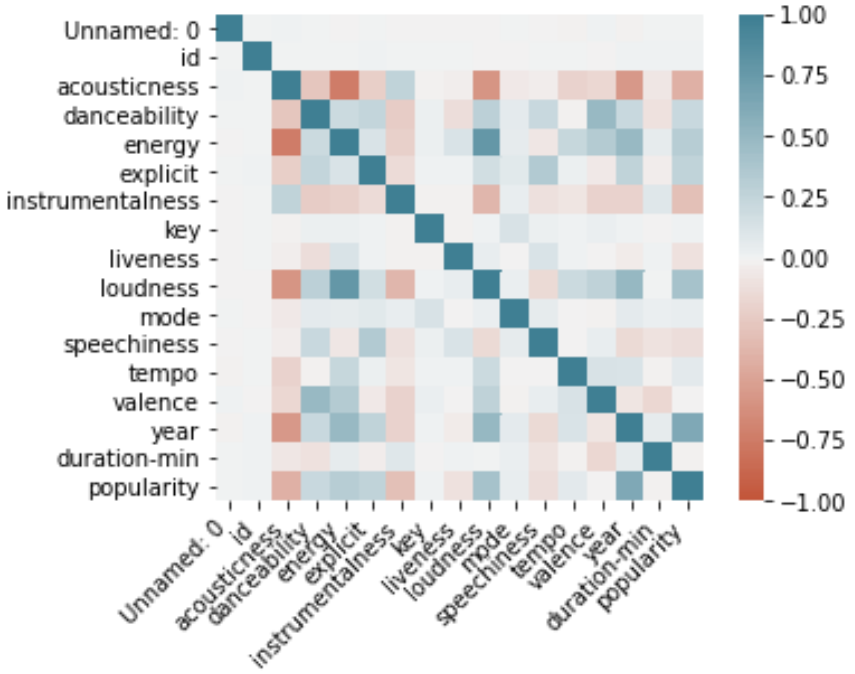
Mode of track: Minor or Major

Month wise release of tracks



Most of tracks were released in January month.

Correlation between different columns of dataset



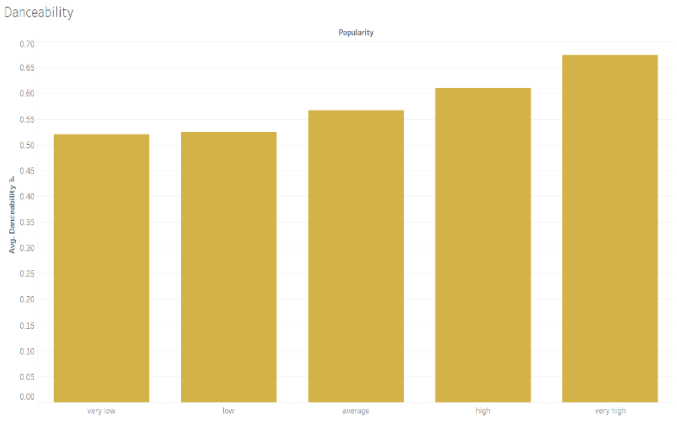
Fact

From this correlation matrix, it's evident that energy, loudness and year are very positively correlated with popularity that is higher their value, higher is the popularity. While Accousticness, Instrumentalness are very negatively correlated with popularity.

Correlation with popularity

Column Name	Correlation
Accousticness	-0.407964
Instrumentalness	-0.317268
Speechiness	-0.122614
Liveness	-0.103116
Mode	-0.039218
Duration-min	-0.009381
Valence	-0.005329
Tempo	0.076751
Danceability	0.220554
Explicit	0.260153
Energy	0.327218
Loudness	0.413219
Year	0.635493
Popularity	1.000000

Column wise analysis

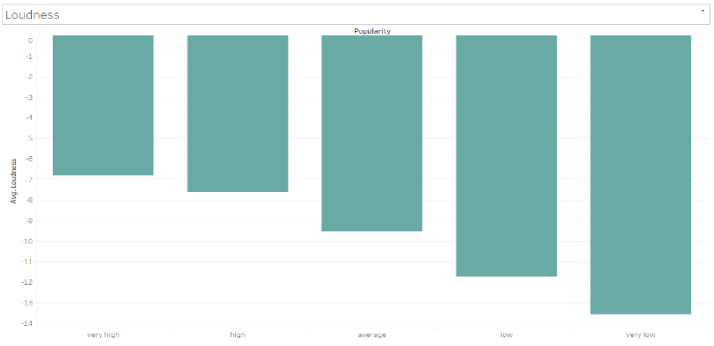


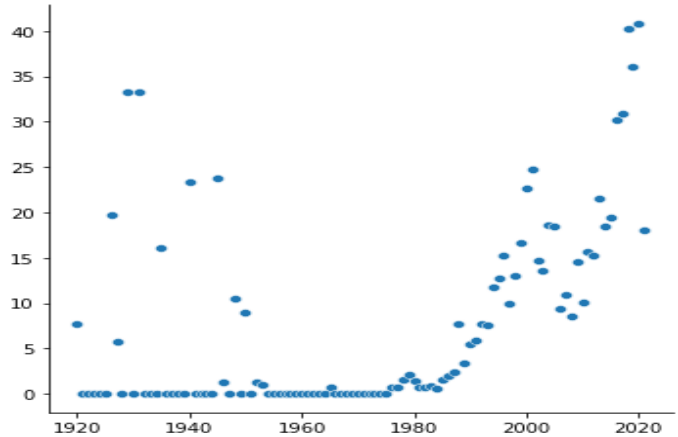
Danceability Vs Popularity

As evident from bar plot that on an average very highly popular songs have high danceability which is very well justified by the fact that people generally play these danceable songs very often in parties, occasions and festivals so more danceable songs gain better popularity as compared to less danceable songs

Loudness Vs Popularity

People prefer loud songs! Yeah, loud songs are very popular. People generally prefer songs with good volume and pitch this creates energy, zeal and positive vibes in them.

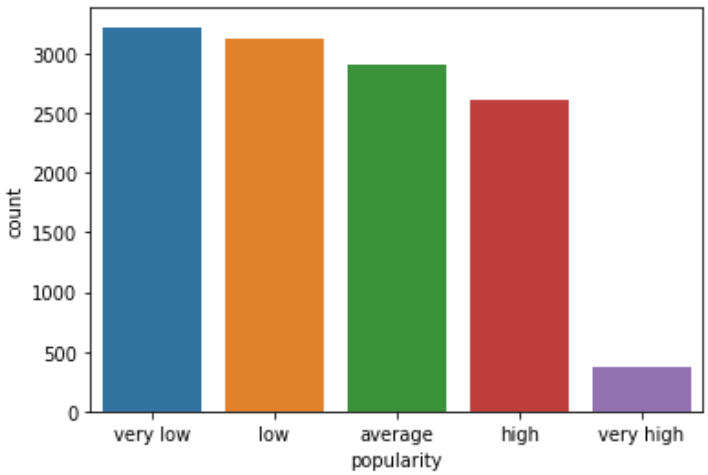




Year wise percentage explicitness

As we can see that trend shifts towards more explicit tracks from non-explicit tracks and is increasing very rapidly. But here you may see some anomalies this is because Dirty Blues Band which was famous during the era 1920-1970 released many explicit tracks and so people loved that therefore that caused anomalies in general trend.

Distribution of Popularity

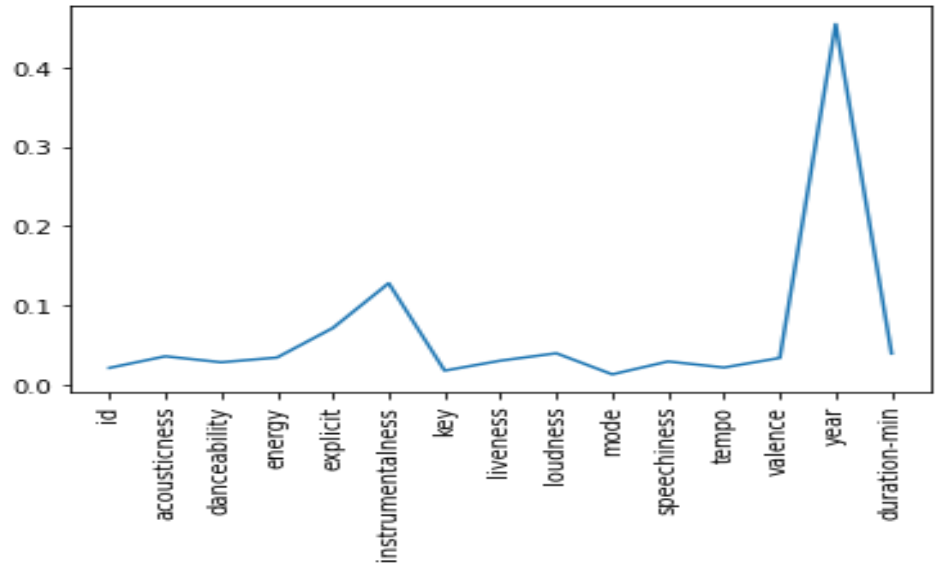


Our target variable "popularity" is not uniformly distributed and its very skewed. If we use such dataset for training our Machine learning model then it will produce biased results and give less importance to less frequent class. So balancing is required. To avoid this, we preprocess our data and apply Synthetic Minority Oversampling Technique known as SMOTE. This will balance all classes and makes new instances for minority class by oversampling. See appendix for more information

Decade wise importance of features in predicting popularity

Decade	Important Feature
1920-30	Accousticness
1930-40	Accousticness
1940-50	key
1950-60	loudness
1960-70	loudness
1970-80	loudness
1980-90	danceability
1990-2000	loudness
2000-10	loudness
2010-20	danceability

Feature Importance



"Year" feature got the most importance and as it was positively correlated with popularity depicted that newer song has higher popularity than old ones. Also, its quite obvious that people's taste changes with time and new songs are adapting their taste resulting into better popularity. Also next important factor was Instrumentalness but it was negatively correlated so higher it is, lower the popularity.

Feature Engineering

Binning of Months in seasons

Many songs are considered seasonal in a sense that during a particular season if released then its popularity can be maximized. We made a new feature by binning some months to make a season and hence some particular types of songs can be correlated with a season too. For example, if a few songs with a certain set of features were popular with only their season as a common point or one of the few common points then season could be helpful in grouping 'similar' songs together and hence assign similar popularity to them. It was observed that adding this feature did indeed increase our accuracy.

Label encoding

Encoded categorical variables like string into numerical data types to make it fit for machine learning models to train upon.

Popularity: - Very low –> 1, low –> 2, average --> 3, high --> 4, very high --> 5

One Hot Encoding

Makes as many Boolean column as number of categories that represent 1 if that that instance belongs to that class and 0 for the rest of all.

Explicit: Yes --> 1, No --> 0

Mode: - Major --> 1, Minor --> 0

Normalization

We converted various distribution to normal distribution having mean at zero and variance equal to 1. This scales data to proper distribution fit for model to train.

Data Preprocessing

Cleaning Dataset – removing outliers

Data splitting

We Split **70%** dataset into train-set and **30%** into validation-set**.** The training set is used to train our model. We have the validation set in order to evaluate our model's prediction. We split dataset using Sklearn Train Test split function that shuffles data and randomly partitions into train set and test set (here validation set)

- X_train is the training data set.
- y_train is the set of output to all the data in X_train.
- X_val is the validation set.
- y_val is the set of output to all data in X_val

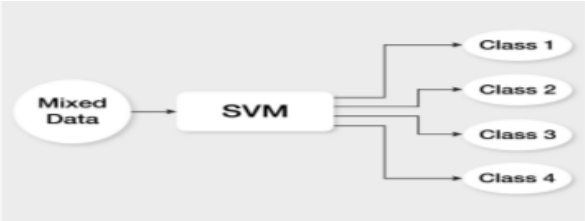
Over sampling dataset

In order to make model bias free, we synthetically increase the instances of minority class (here “very high” popularity class) and made popularity distribution uniform. For this we used **SMOTE** package that created new instance from previous instances.

Model Selection: -

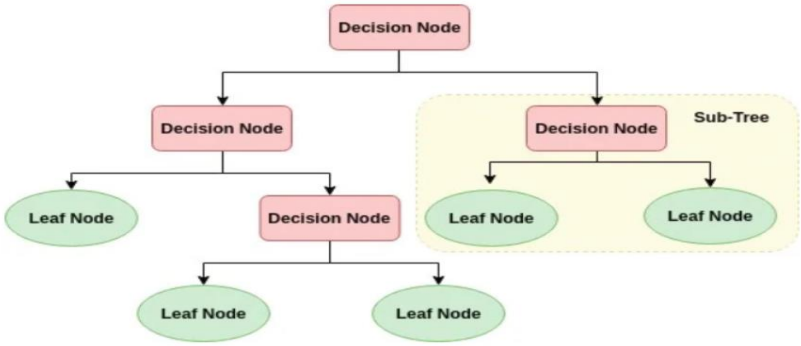
Support Vector Machine (SVM)

A supervised machine learning algorithm which can be used for both classification or regression challenges. The idea of SVM is simple: The algorithm creates a line (or a hyperplane) which separates the data into classes.



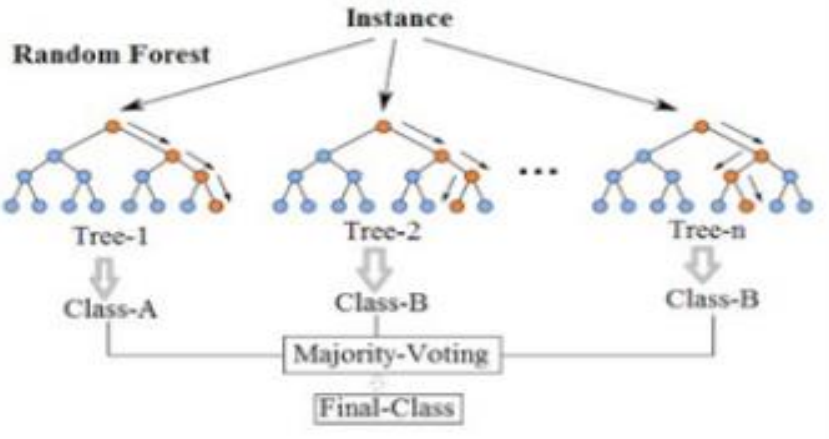
Decision Tree classifier

Decision tree learning is one of the predictive modelling approaches used in statistics, data mining and machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves).



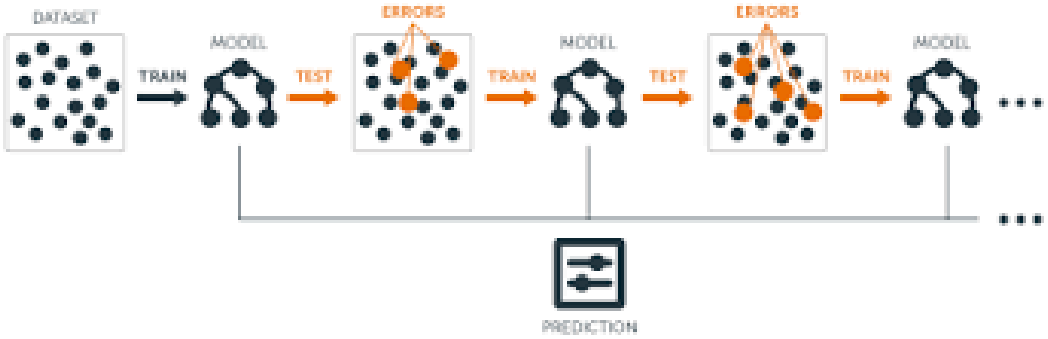
Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.



Gradient boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models typically decision trees.

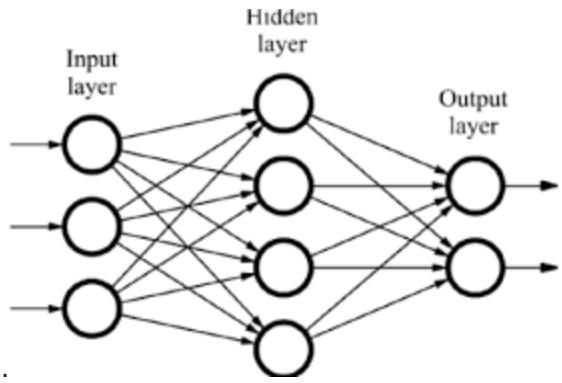


K-Nearest Neighbours

K nearest neighbours is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

Neural Network

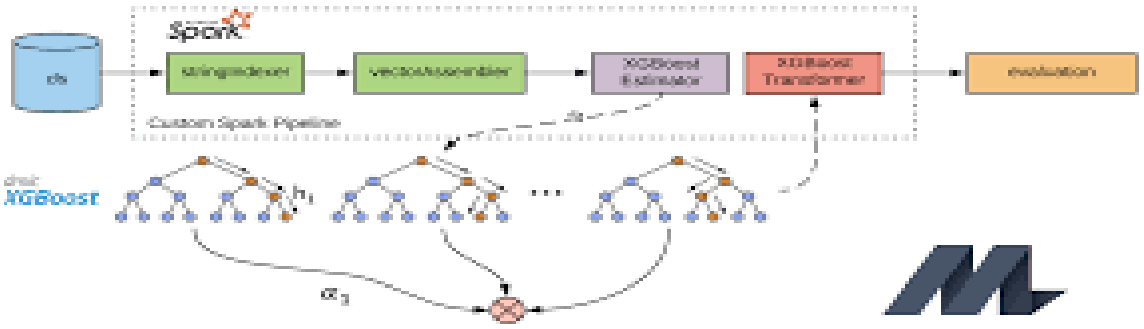
A Neural Network is a computational learning system that uses a network of functions to understand and translate a data input of one form into a desired output, usually in another form. Essentially input is multiplied by specific weights and transformed to our desired output over one or multiple such layers of weights



weights.

XGBoost

XGBoost is a decision-tree-based ensemble algorithm that uses a gradient boosting framework. It is basically an optimized gradient boosting method where each new tree is based on the previous tree created



Model selection

Model Name	Accuracy	Balanced accuracy	Kappa	F1
Decision tree	**0.59**	**0.59**	**0.49**	**0.59**
Random forest	**0.71**	**0.71**	**0.64**	**0.71**
Gaussian naïve bayes	**0.5**	**0.49**	**0.37**	**0.5**
Gradient boost	**0.65**	**0.65**	**0.57**	**0.65**
Adaboost	**0.58**	**0.58**	**0.58**	**0.48**
XGBoost	**0.74**	**0.70**	**0.67**	**0.72**
Neural network	**0.69**	**0.68**	**0.62**	**0.69**
Logistic regression	**0.53**	**0.52**	**0.41**	**0.53**
SVM classifier	**0.64**	**0.64**	**0.55**	**0.64**
KNN classifier	**0.60**	**0.60**	**0.50**	**0.60**
NN + Random forest	**0.71**	**0.70**	**0.638**	**0.71**

Evaluation metrics:

Categorical Accuracy : Categorical Accuracy calculates the percentage of predicted values (yPred) that match with actual values (yTrue) for one-hot labels. For a record: We identify the index at which the maximum value occurs using argmax(). If it is the same for both yPred and yTrue, it is considered accurate.

Custom metric: We have also developed a custom metric which is :-

M = Correct_bid + 0.25*Overvalued_bid - Undervalued_bid;

This is of the above form because we wanted our model to predict either correct bid or higher than correct bid so songs don't get untraded.

Correct_bid = Percentage of correct predictions.

Overvalued_bid = Percentage of predictions where predictions is higher than actual bid.

Undervalued_bid = Percentage of predictions where predictions is lower than actual bid.

Hyperparameter Tuning

Hyperparameter	Value / Argument
Xgboost	**(**base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.3, gamma=0.0, gpu_id=-1, importance_type='gain', interaction_constraints='', learning_rate=0.25, max_delta_step=0, max_depth=4, min_child_weight=5, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=8, num_parallel_tree=1, objective='multi:softprob', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None **)**

Conclusion

We used all the above methods and got best accuracy from XGBoost method.

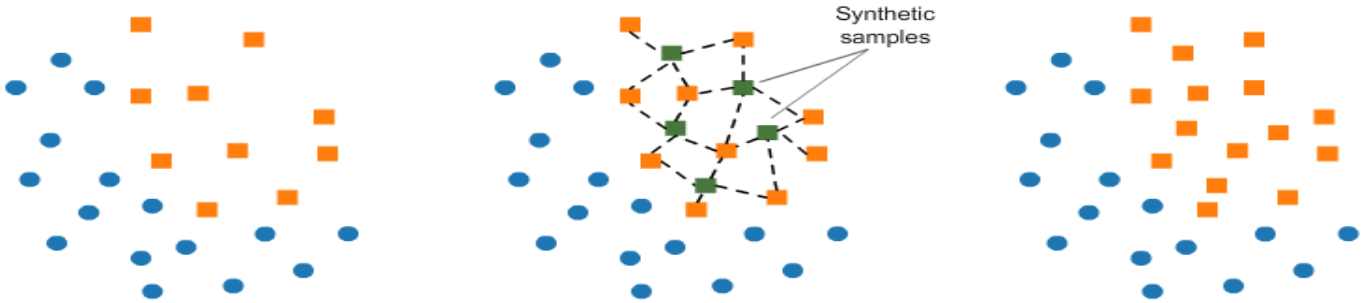
Accuracy – 74 %
Balanced accuracy – 70%
Kappa – 0.67
F1 – 0.72

Annexure : APPENDIX

SMOTE (Synthetic Minority Oversampling Technique)

https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

This technique was described by Nitesh Chawla, et al. in their 2002 paper named for the technique titled "SMOTE: Synthetic Minority Over-sampling Technique."



This approach involves duplicating examples in the minority class, although these examples don't add any new information to the model. Instead, new examples can be synthesized from the existing examples. This is a type of data augmentation for the minority class. SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbours for that example are found (typically k=5). A randomly selected neighbour is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

Converting categorical variables into numerical values

1. Label Encoding

Label Encoding refers to converting the labels into numeric form so as to convert it into the machine-readable form. This even helps to carry out any numerical transformation.

Syntax

```
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
df['species']= label_encoder.fit_transform(df['species'])
```

2. One-Hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. Just label encoding is not sufficient to provide to the model for training because as the number of unique entries increases, the categorical values also proportionally increases. The higher the categorical value it assumes , better the category for the model. This could lead to inaccuracies and wrong results.

Syntax:

```
onehot_encoder  = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
```

References: -

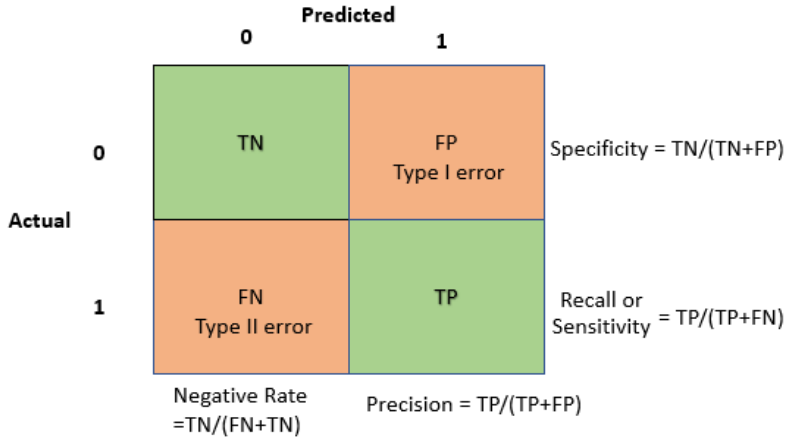
Pandas: offers data structures and operations for manipulating numerical tables and time series.
Documentation: https://pandas.pydata.org/pandas-docs/stable/

NumPy: used for computing scientific and mathematical data like numerical and linear analysis
Documentation: https://numpy.org/doc/

Matplotlib: is a plotting library for the Python and its numerical mathematics extension NumPy.
Documentation: https://matplotlib.org/3.1.1/contents.html

Sklearn: provides functions for various machine learning models and evaluation metrices
Documentation**:** https://scikit-learn.org/

Evaluation Metrices



$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

$$\text{F1 - Score} = \frac{2*Recall*Precision}{Recall + Precision}$$

Balanced Accuracy = (Sensitivity + Specificity) / 2

Kappa

$$K = \frac{P_{observed} - P_{chance}}{1 - P_{chance}}$$