MINOR PROJECT -1

ODD SEMESTER 2020

MUSIC PREDICTION MODEL

Enrolment Number: 18102257

Name: Syed Rahim Ali

Faculty Supervisor: Mr. Ritesh Sharma

Submitted in partial fulfilment of the Degree of

Bachelor of Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION

JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY

CONTENTS

CERTIFICATE

This is to certify that the work titled "**Music Prediction Model** " submitted by **Syed Rahim Ali** in partial fulfilment for the award of degree of Bachelor of Technology of Jaypee Institute of Information Technology, Noida has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

Signature of Supervisor …………………...

Name of Supervisor: Mr. Ritesh Sharma

Designation: Assistant Professor

Date:

ACKNOWLEDGEMENT

I would like to express my special gratitude to the Electronics and Communications Department of Jaypee Institute of Information and Technology for giving me the golden opportunity to undertake this project. My sincere thanks to my faculty supervisor, **Mr. Ritesh Sharma** for guiding me throughout the project and helping me improvise and learn so much.

Signature of the student …………………...

Name of the Student: **Syed Rahim Ali**

Date:

SUMMARY

We will use Spotify API to download the song features of our personal playlists and that will serve as our dataset for this project

Since music is no longer exclusive to any music streaming service and the large number of music streaming applications available such as Apple Music, Spotify, Gaana, Saavn etc.

One of the most important features of a successful service is music recommendation algorithms of a service, hence the need for predicting the music taste of a user and suggesting songs accordingly becomes important. Throughout this project we will reiterate how music taste is subjective rather than objective but, nevertheless we will try to find the best results we can using various algorithms.

This project will primarily use functions Sklearn library. We will perform a comparative study of all the Machine Learning algorithms that will be used, and which yield the best result. We will use a wide range of performance measure such as Precision, Recall and ROC-AUC for better understanding.

<u>Chapter 1:</u> <u>Introduction</u>

Rapid development of mobile devices and internet has made possible for us to access different music resources freely. The number of songs available exceeds the listening capacity of single individual. People sometimes feel difficult to choose from millions of songs. Moreover, music service providers need an efficient way to manage songs and help their costumers to discover music by giving quality recommendation. Thus, there is a strong need of a good recommendation system.

In this project we will use Spotify API to collect music data of a user. Two playlists will be needed, one of the songs liked by the user and the other of the disliked songs.

Spotify API gives users the ability to download a song’s audio features. These features include attributes such as a song’s tempo, level of acousticness, and how danceable a song is.

Chapter 2: Data Attributes

Here is a list of all the song features that will be used for the classification task:

Feature	Data Type	Description
DURATION_MS	int	The duration of the track in milliseconds.
KEY	int	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
MODE	int	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
TIME_SIGNATURE	int	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
ACOUSTICNESS	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
DANCEABILITY	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
ENERGY	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has

		high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.
INSTRUMENTALNESS	float	Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
LIVENESS	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
LOUDNESS	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
SPEECHINESS	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music.

		Values below 0.33 most likely represent music and other non-speech-like tracks.
VALENCE	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
TEMPO	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.
ID	string	The Spotify ID for the track.
CLASS	int	Class = 1 if user liked the song and class = 0 If the user disliked the song.

Chapter 3: Data Selection & Data Cleaning

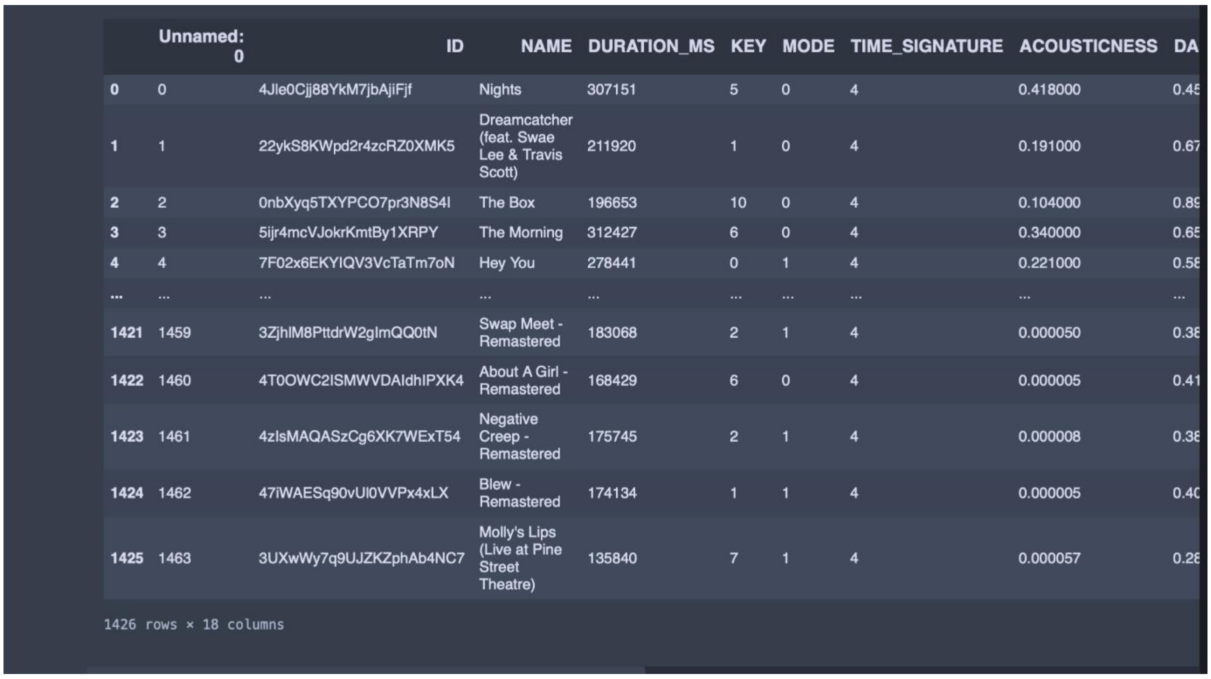
3.1 Data Selection

One of the most time consuming in this task was the preparation of the playlists for the dataset. We used both of our playlists for this task. One for ‘Liked’ and the other for ‘Disliked’. Creating the ‘Disliked’ playlist can be the trickier than it seems, we could just randomly select songs from the 70’s and 80’s or songs from an entirely different language to create this playlist. However, to minimize bias, the ‘Disliked’ playlist had to be as diverse as the ‘Liked’ one.

In case of music recommendation systems, we cannot be too specific while selecting the songs. Liking or disliking music is a subconscious process, we must prepare the data the same way.

3.2 Data Cleaning

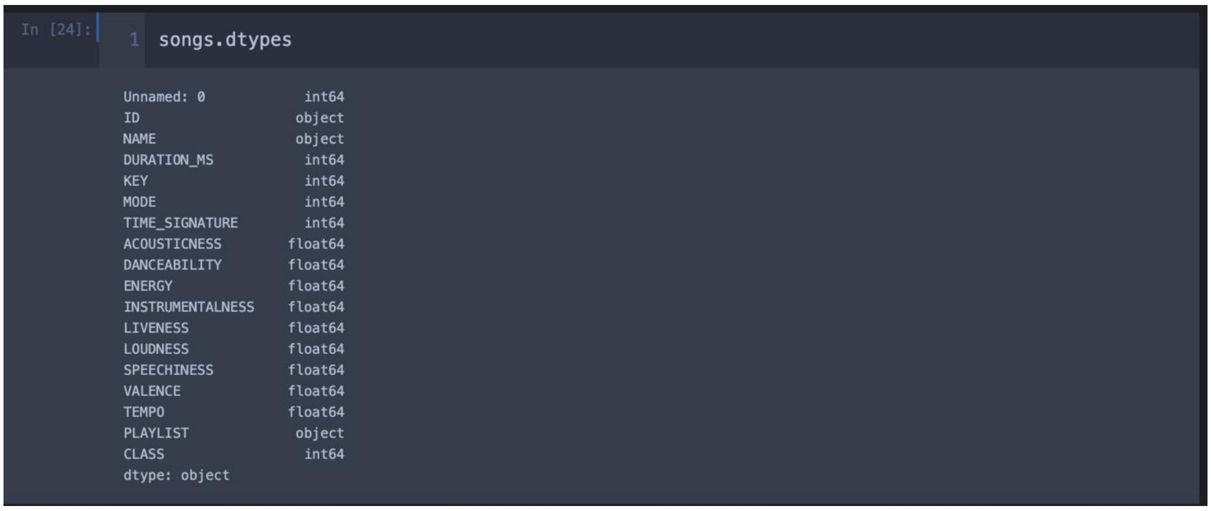
After downloading the data csv file use Pandas library to download the file into the Jupyter Notebook.



	Unnamed: 0	ID	NAME	DURATION_MS	KEY	MODE	TIME_SIGNATURE	ACOUSTICNESS	DA
0	0	4Jle0Cjj88YkM7jbAjiFjf	Nights	307151	5	0	4	0.418000	0.45
1	1	22ykS8KWpd2r4zcRZ0XMK5	Dreamcatcher (feat. Swae Lee & Travis Scott)	211920	1	0	4	0.191000	0.67
2	2	0nbXyq5TXYPCO7pr3N8S4l	The Box	196653	10	0	4	0.104000	0.89
3	3	5ijr4mcVJokrKmtBy1XRPY	The Morning	312427	6	0	4	0.340000	0.65
4	4	7F02x6EKYIQV3VcTaTm7oN	Hey You	278441	0	1	4	0.221000	0.58
...	...	...	...	...	...	...	...	...	...
1421	1459	3ZjhlM8PttdrW2glmQQ0tN	Swap Meet - Remastered	183068	2	1	4	0.000050	0.38
1422	1460	4T0OWC2ISMWVDAldhIPXK4	About A Girl - Remastered	168429	6	0	4	0.000005	0.41
1423	1461	4zIsMAQASzCg6XK7WExT54	Negative Creep - Remastered	175745	2	1	4	0.000008	0.38
1424	1462	47iWAESq90vUI0VVPx4xLX	Blew - Remastered	174134	1	1	4	0.000005	0.40
1425	1463	3UXwWy7q9UJZKZphAb4NC7	Molly's Lips (Live at Pine Street Theatre)	135840	7	1	4	0.000057	0.28

```
1426 rows × 18 columns
```

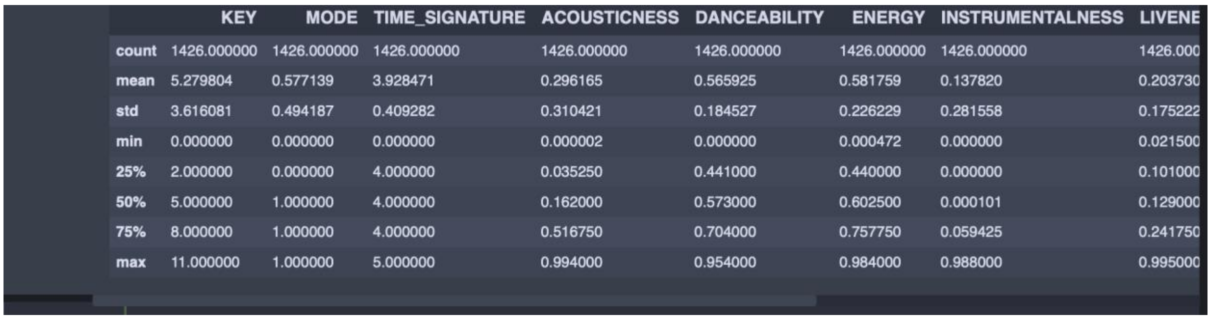
Figure 1 DataFrame



```
In [24]:  songs.dtypes

Unnamed: 0           int64
ID                  object
NAME                object
DURATION_MS          int64
KEY                  int64
MODE                 int64
TIME_SIGNATURE       int64
ACOUSTICNESS       float64
DANCEABILITY       float64
ENERGY             float64
INSTRUMENTALNESS   float64
LIVENESS           float64
LOUDNESS           float64
SPEECHINESS        float64
VALENCE            float64
TEMPO              float64
PLAYLIST            object
CLASS                int64
dtype: object
```

Figure 2 Attribute Datatypes

Looking at all the attributes of the dataset we can observe that it contains some attributes are not relevant to the prediction process(‘Unnamed: 0’, ‘ID’, ‘Name’, ‘Duration_ms’, ‘ Playlist’).So we will drop these columns.



	KEY	MODE	TIME_SIGNATURE	ACOUSTICNESS	DANCEABILITY	ENERGY	INSTRUMENTALNESS	LIVENE
count	1426.000000	1426.000000	1426.000000	1426.000000	1426.000000	1426.000000	1426.000000	1426.000
mean	5.279804	0.577139	3.928471	0.296165	0.565925	0.581759	0.137820	0.203730
std	3.616081	0.494187	0.409282	0.310421	0.184527	0.226229	0.281558	0.175222
min	0.000000	0.000000	0.000000	0.000002	0.000000	0.000472	0.000000	0.021500
25%	2.000000	0.000000	4.000000	0.035250	0.441000	0.440000	0.000000	0.101000
50%	5.000000	1.000000	4.000000	0.162000	0.573000	0.602500	0.000101	0.129000
75%	8.000000	1.000000	4.000000	0.516750	0.704000	0.757750	0.059425	0.241750
max	11.000000	1.000000	5.000000	0.994000	0.954000	0.984000	0.988000	0.995000

Figure 3 Summary of Attributes

Summary of all numerical attributes of the data structure.

Chapter 4: Splitting Data

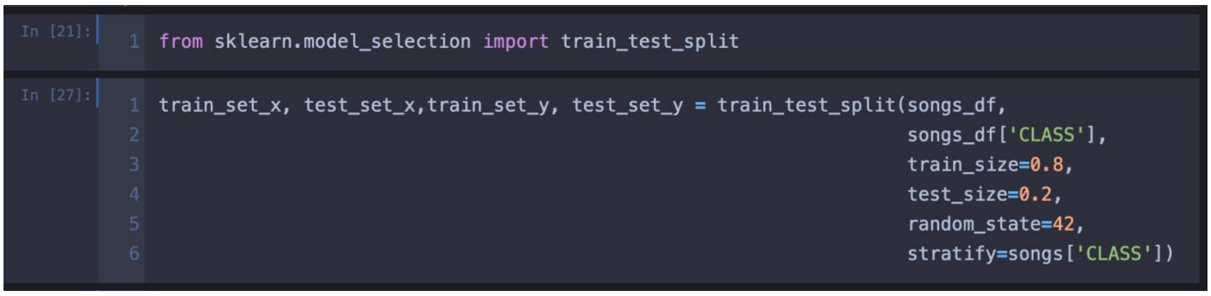
After cleaning the data of any unwanted attributes, we split the dataset into ‘Training Data’ and ‘Testing Data’.

4.1 Training Data

The observations in the training set form the experience that the algorithm uses to learn. In supervised learning problems, each observation consists of an observed output variable and one or more observed input variables

4.2 Test Data

The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.



```
In [21]:  from sklearn.model_selection import train_test_split

In [27]:  train_set_x, test_set_x,train_set_y, test_set_y = train_test_split(songs_df,
                                                                        songs_df['CLASS'],
                                                                        train_size=0.8,
                                                                        test_size=0.2,
                                                                        random_state=42,
                                                                        stratify=songs['CLASS'])
```

Figure 4 Train-Test Split

While splitting the dataset it is important to stratify the data with the target variables. Stratification is done so that the training data and testing data have a proportional distribution of the target values, i.e. no of liked and disliked songs in both testing and training dataset will almost be equal. This reduces bias while training dataset.

Chapter 5: Exploratory Data Analysis

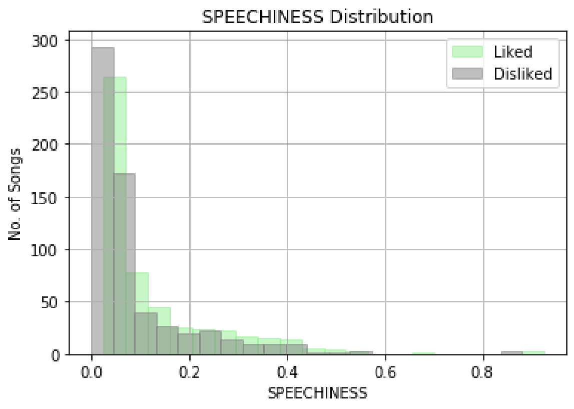


Figure 5 Speechiness

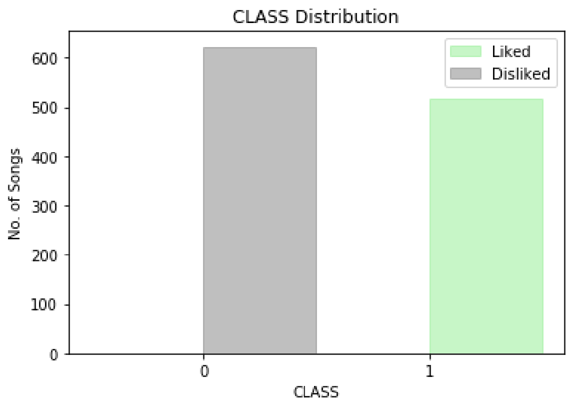


Figure 6 Class

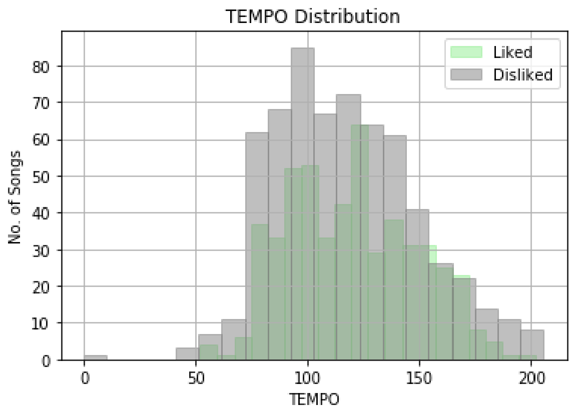


Figure 7 Tempo

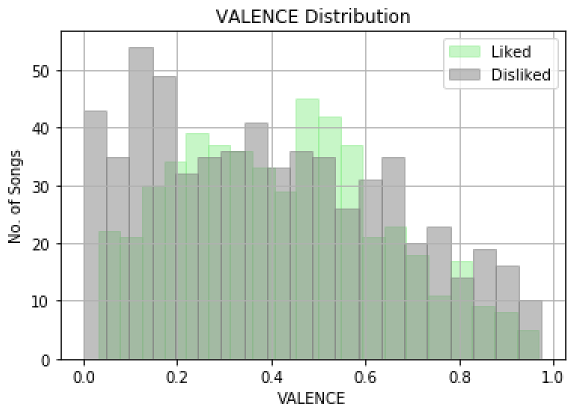


Figure 8 Valence

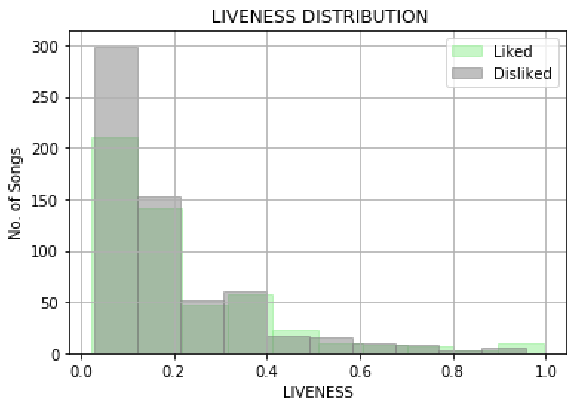


Figure 9 Liveness

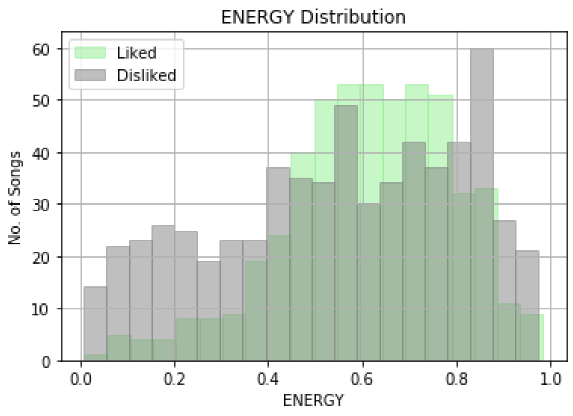


Figure 10 Energy

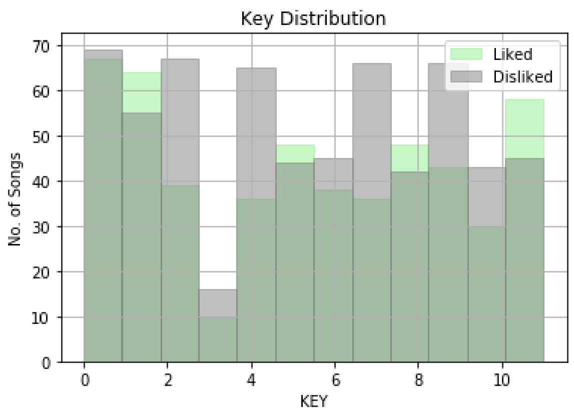


Figure 11 Key

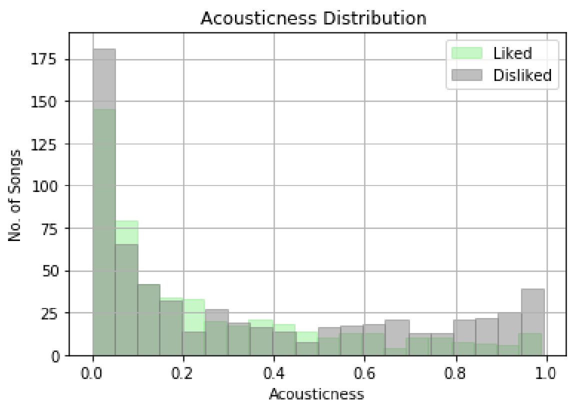


Figure 12 Acousticness

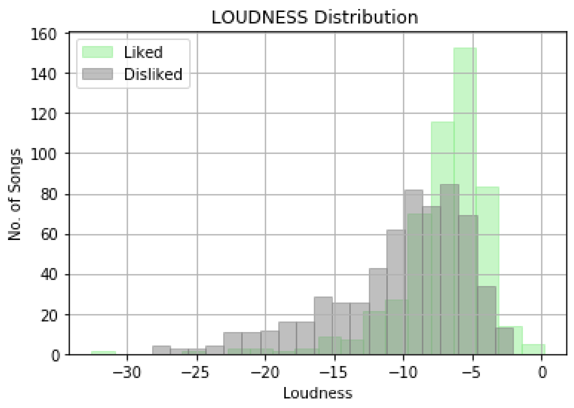


Figure 13 Loudness

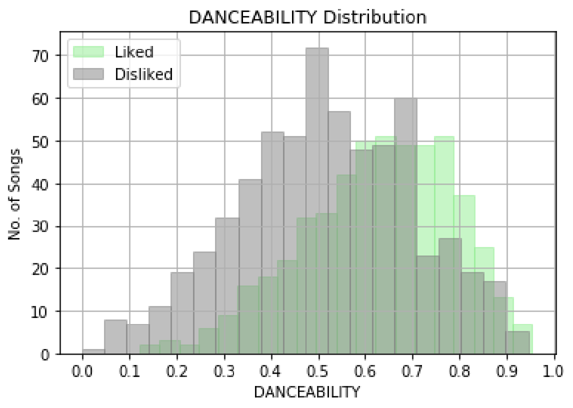


Figure 14 Danceability

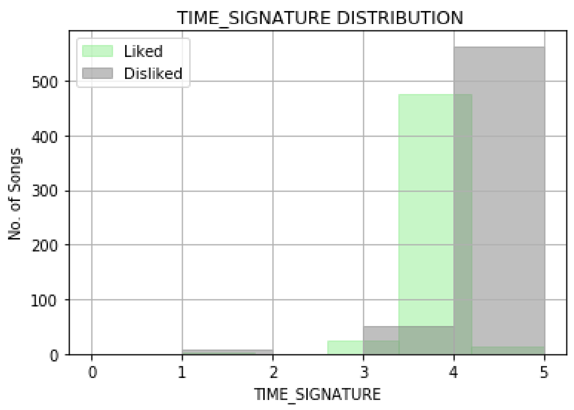


Figure 15 Time Signature

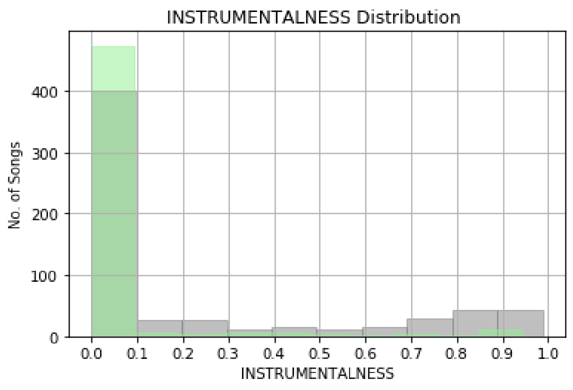


Figure 16 Instrumentalness

5.1 Observations from EDA

- These plots give us a general idea of the distribution of various features and their concentration.
- From the distribution of Danceability, Loudness and Energy slight correlations can be observed.
- We observed just an iota of correlation between the selected features and the target variable. This meant that my task of devising a competent classifier was a difficult one and that a linear classifier would most likely not be sufficient.

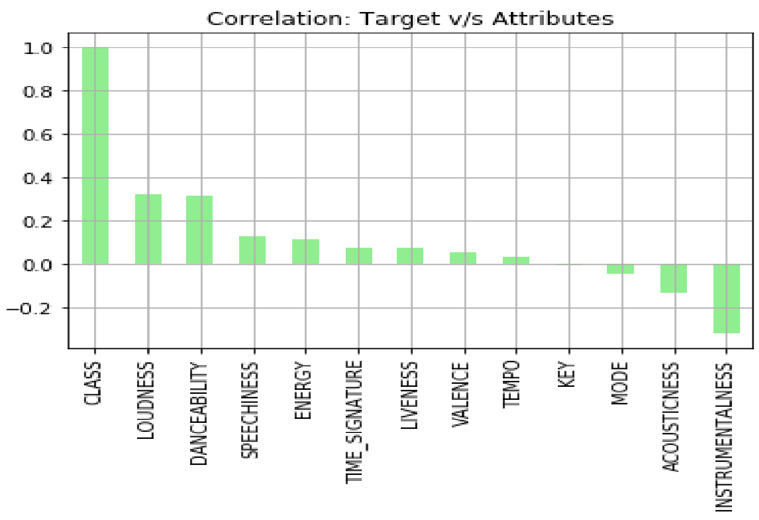


Figure 17 Correlation: Target v/s Attributes

- Features like Time Signature, Valence, Tempo, Key, etc. are not relevant.

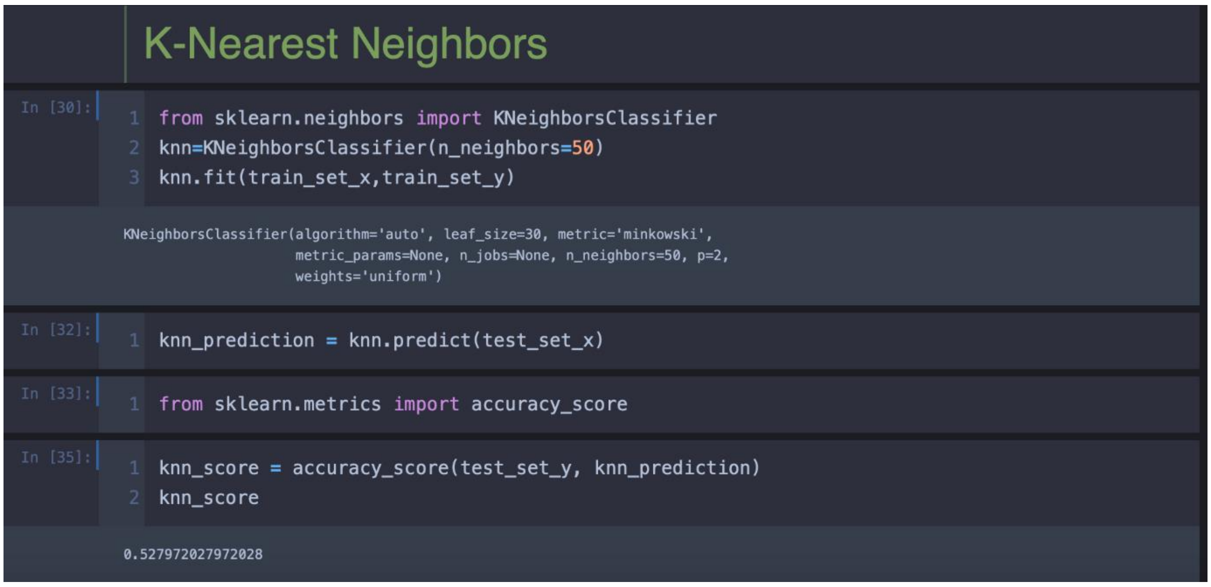
Chapter 6: Application of Machine Learning Algorithms

For training our data we will use 6 classification algorithms from Sklearn.

- K-Nearest Neighbors
- Decision Tree
- Stochastic Gradient Descent
- Support Vector Classifier
- Random Forest
- XGBoost

6.1 K-Nearest Neighbours

K Nearest Neighbor(KNN) is a very simple, easy to understand, versatile and one of the topmost machine learning algorithms. KNN is a non-parametric and lazy learning algorithm. Non-parametric means there is no assumption for underlying data distribution. In other words, the model structure determined from the dataset. This will be very helpful in practice where most of the real-world datasets do not follow mathematical theoretical assumptions. Lazy algorithm means it does not need any training data points for model generation. All training data used in the testing phase. This makes training faster and testing phase slower and costlier. Costly testing phase means time and memory. In the worst case, KNN needs more time to scan all data points and scanning all data points will require more memory for storing training data.



K-Nearest Neighbors

```
In [30]: from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=50)
         knn.fit(train_set_x,train_set_y)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=50, p=2,
                     weights='uniform')

In [32]: knn_prediction = knn.predict(test_set_x)

In [33]: from sklearn.metrics import accuracy_score

In [35]: knn_score = accuracy_score(test_set_y, knn_prediction)
         knn_score

0.527972027972028
```

Figure 18 K-Nearest Neighbor Implementation

6.2 Decision Tree

Decision Tree Classifier: A Decision Tree Classifier is a type of Tree based algorithm i.e. it uses decision tree to classify observations.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

Some advantages of decision trees are:

- Simple to understand and to interpret. Trees can be visualised.
- Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

The disadvantages of decision trees include:

- Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning, setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms

cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

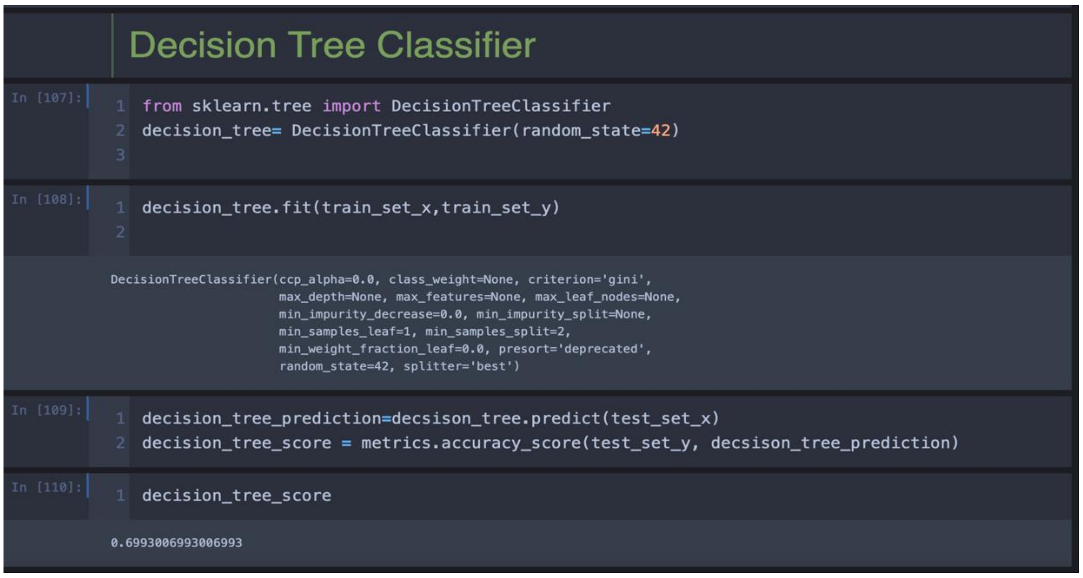
Decision Tree is a simple representation for classifying examples. It is a Supervised Machine Learning where the data is continuously split according to a certain parameter.

Decision Tree consists of:

1. **Nodes**: Test for the value of a certain attribute.
2. **Edges/ Branch**: Correspond to the outcome of a test and connect to the next node or leaf.
3. **Leaf nodes**: Terminal nodes that predict the outcome (represent class labels or class distribution).

Advantages of Classification with Decision Trees:

1. Inexpensive to construct.
2. Extremely fast at classifying unknown records.
3. Easy to interpret for small-sized trees
4. Accuracy comparable to other classification techniques for many simple data sets.
5. Excludes unimportant features.


Decision Tree Classifier

```
In [107]:
from sklearn.tree import DecisionTreeClassifier
decision_tree= DecisionTreeClassifier(random_state=42)
```

```
In [108]:
decision_tree.fit(train_set_x,train_set_y)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

```
In [109]:
decision_tree_prediction=decsison_tree.predict(test_set_x)
decision_tree_score = metrics.accuracy_score(test_set_y, decsison_tree_prediction)
```

```
In [110]:
decision_tree_score
```

```
0.6993006993006993
```

Figure 19 Decision Tree Implementation

Disadvantages of Classification with Decision Trees:

1. Easy to overfit.
2. Decision Boundary restricted to being parallel to attribute axes.
3. Decision tree models are often biased toward splits on features having many levels.
4. Small changes in the training data can result in large changes to decision logic.
5. Large trees can be difficult to interpret and the decisions they make may seem counter intuitive.

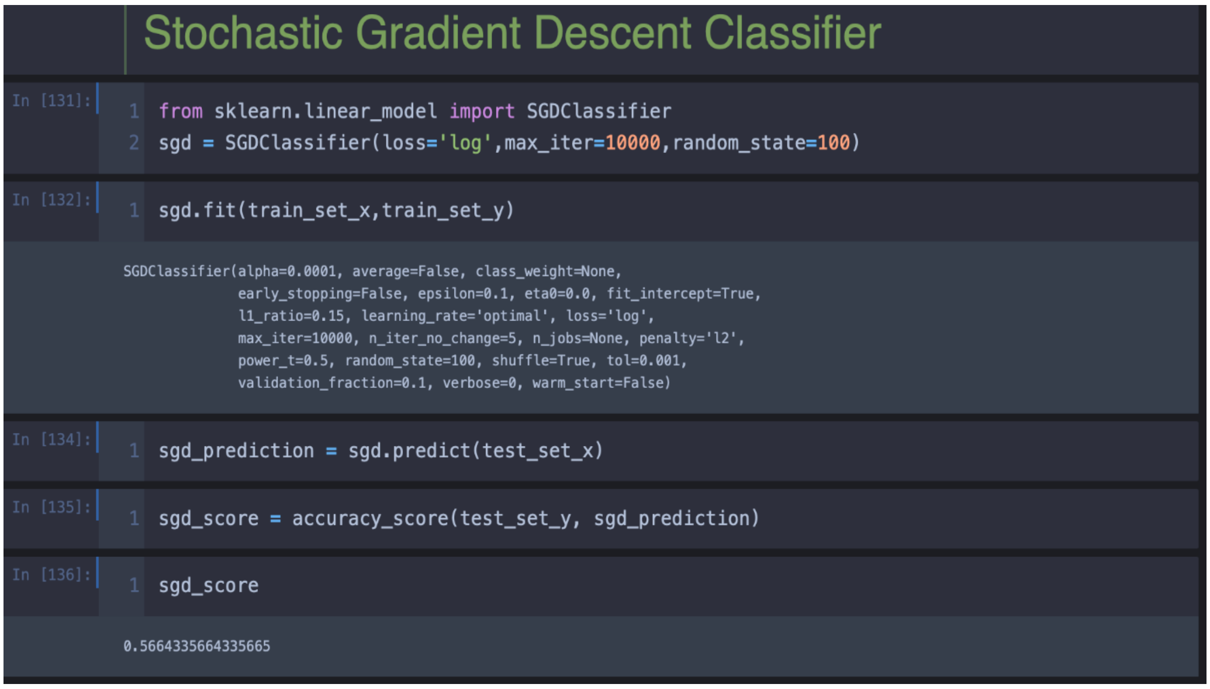
6.3 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. Even though SGD has been around in the machine learning community for a long time, it has received a considerable amount of attention just recently in the context of large-scale learning.

The advantages of Stochastic Gradient Descent are:

- Efficiency.
- Ease of implementation (lots of opportunities for code tuning).

Using SGD Classifier(loss='log') results in logistic regression, i.e. a model equivalent to Logistic Regression which is fitted via SGD.



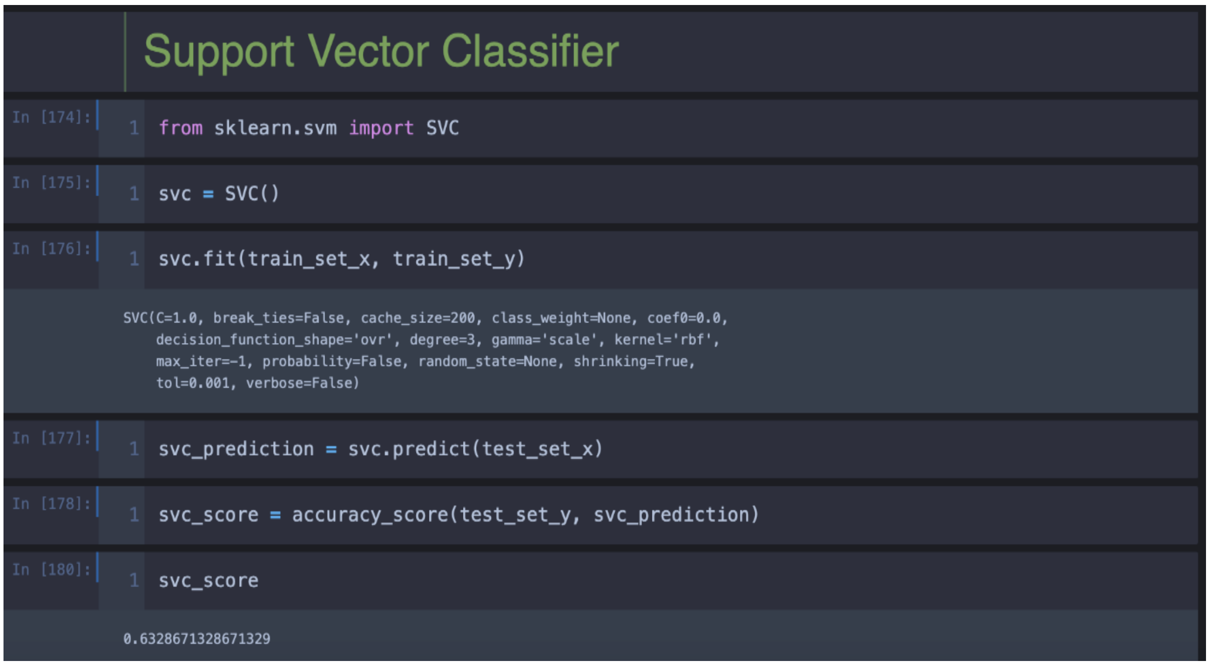
Stochastic Gradient Descent Classifier

```
In [131]: from sklearn.linear_model import SGDClassifier
          sgd = SGDClassifier(loss='log',max_iter=10000,random_state=100)

In [132]: sgd.fit(train_set_x,train_set_y)

SGDClassifier(alpha=0.0001, average=False, class_weight=None,
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log',
              max_iter=10000, n_iter_no_change=5, n_jobs=None, penalty='l2',
              power_t=0.5, random_state=100, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)

In [134]: sgd_prediction = sgd.predict(test_set_x)

In [135]: sgd_score = accuracy_score(test_set_y, sgd_prediction)

In [136]: sgd_score

0.5664335664335665
```

Figure 20 SGD Implementation

6.4 Support Vector Classifier

Linear Support Vector Classifier (SVC): supervised machine learning algorithm. The objective of a Linear SVC (Support Vector Classifier) is to fit to the data you provide, returning a "best fit" hyperplane that divides, or categorizes, your data. From there, after getting the hyperplane, you can then feed some features to your classifier to see what the "predicted" class is.



Support Vector Classifier

```
In [174]: from sklearn.svm import SVC

In [175]: svc = SVC()

In [176]: svc.fit(train_set_x, train_set_y)

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)

In [177]: svc_prediction = svc.predict(test_set_x)

In [178]: svc_score = accuracy_score(test_set_y, svc_prediction)

In [180]: svc_score

0.6328671328671329
```

Figure 21 Support Vector Classifier Implementation

6.5 Random Forest Classifier

A random forest is a meta estimator that fits several decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. It is an ensemble method.

The goal of ensemble methods is to combine the predictions of several base estimators built with a given learning algorithm in order to improve generalizability / robustness over a single estimator.

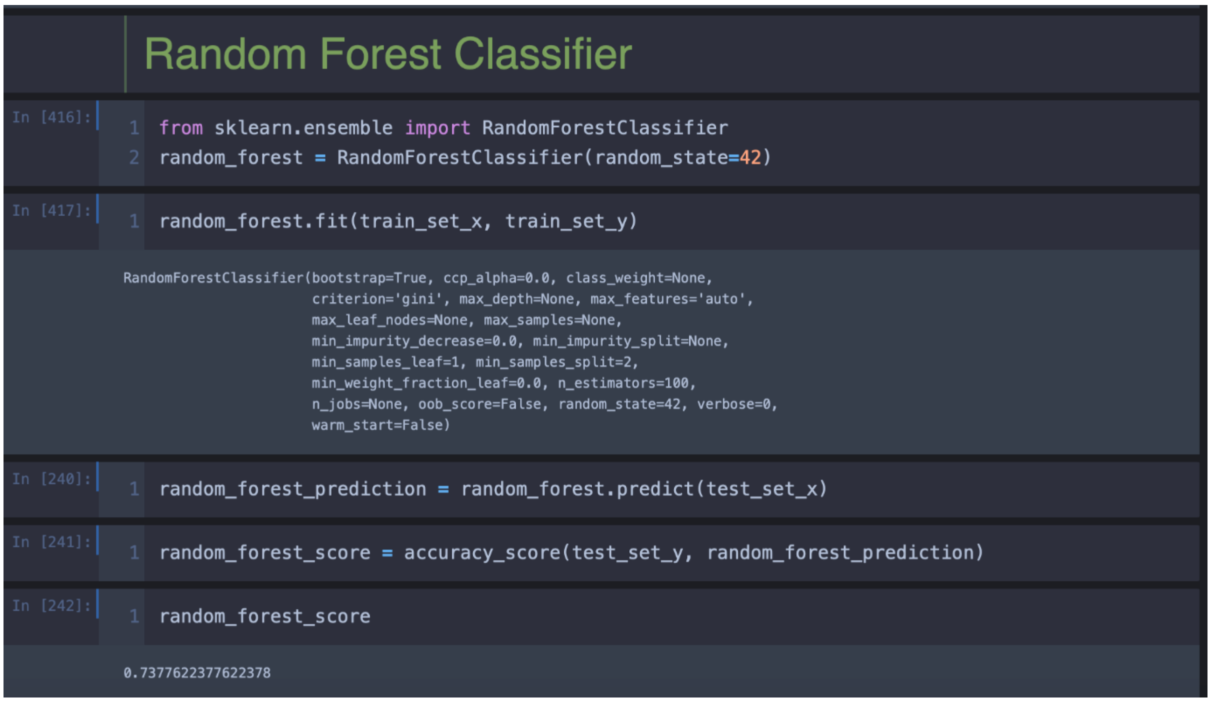
Two families of ensemble methods are usually distinguished:

- In **averaging methods**, the driving principle is to build several estimators independently and then to average their predictions. On average, the combined estimator is usually better than any of the single base estimator because its variance is reduced.

Examples: Bagging methods, Forests of randomized trees, …

- By contrast, in **boosting methods**, base estimators are built sequentially and one tries to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Examples: AdaBoost, Gradient Tree Boosting, …



Random Forest Classifier

```
In [416]:
from sklearn.ensemble import RandomForestClassifier
random_forest = RandomForestClassifier(random_state=42)
```

```
In [417]:
random_forest.fit(train_set_x, train_set_y)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

```
In [240]:
random_forest_prediction = random_forest.predict(test_set_x)
```

```
In [241]:
random_forest_score = accuracy_score(test_set_y, random_forest_prediction)
```

```
In [242]:
random_forest_score
```

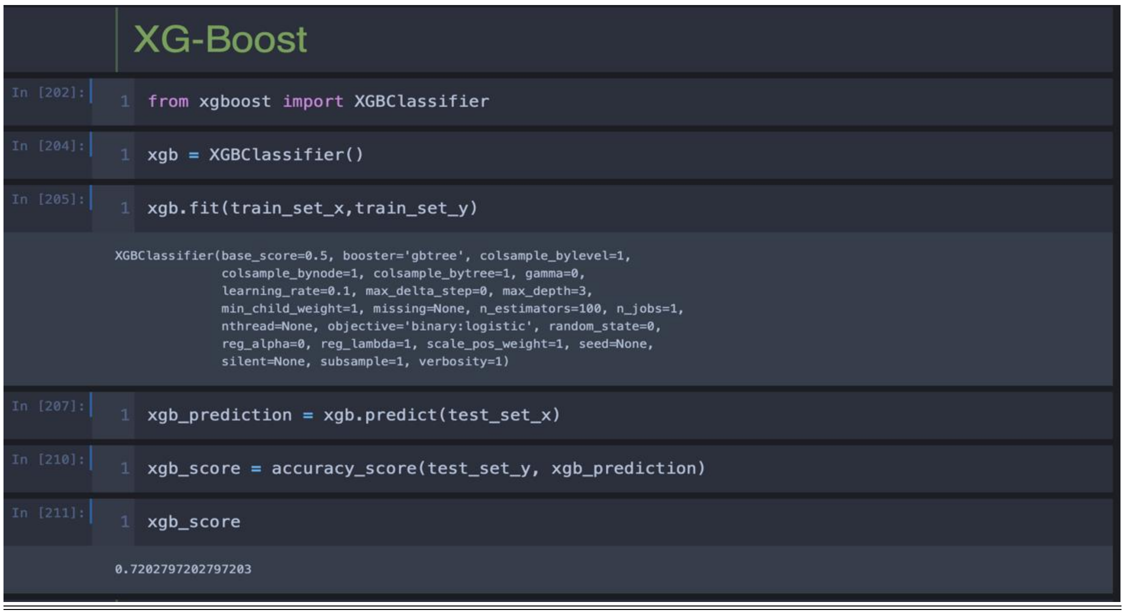
```
0.7377622377622378
```

Figure 22 Random Forest Implementation

6.6 XGBoost Classifier

XGBoost Classifier: The Random Forest Classifier is a part of averaging methods family of ensemble methods, whereas XGBoost Classifier is a part of boosting methods family of ensemble methods**.**

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.



XG-Boost

```
In [202]: from xgboost import XGBClassifier

In [204]: xgb = XGBClassifier()

In [205]: xgb.fit(train_set_x,train_set_y)

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

In [207]: xgb_prediction = xgb.predict(test_set_x)

In [210]: xgb_score = accuracy_score(test_set_y, xgb_prediction)

In [211]: xgb_score

0.7202797202797203
```

Figure 23 XG-Boost Implementation

6.7 Hyperparameter Tuning

A Machine Learning model is defined as a mathematical model with several parameters that need to be learned from the data. By training a model with existing data, we can fit the model parameters.

However, there is another kind of parameters, known as Hyperparameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. Two best strategies for Hyperparameter tuning are:

- **GridSearchCV**
- **RandomizedSearchCV**

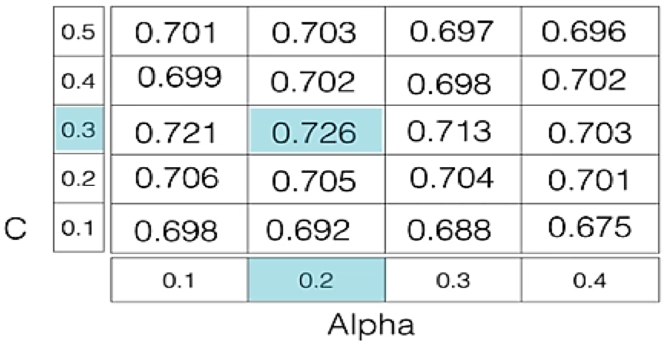
6.7.1 <u>GridSearchCV</u>

In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, because it searches for best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set two hyperparameters C and Alpha of Logistic Regression Classifier model, with different set of values. The grid search technique will construct many versions of the model with all possible combinations of hyperparameters and will return the best one.

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4]. For a combination C=0.3 and Alpha=0.2, performance score comes out to be 0.726(Highest**)**, therefore it is selected.

Drawback: GridSearchCV will go through all the intermediate combinations of hyperparameters which makes grid search computationally very expensive(10+hours).

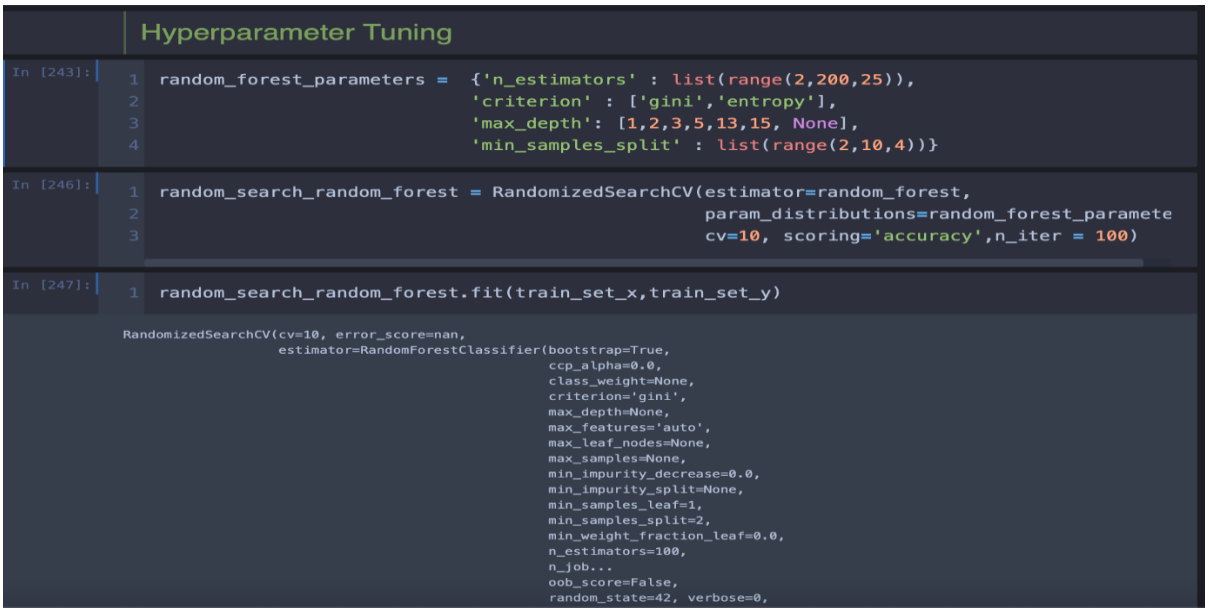


	0.5	0.701	0.703	0.697	0.696
	0.4	0.699	0.702	0.698	0.702
	0.3	0.721	0.726	0.713	0.703
	0.2	0.706	0.705	0.704	0.701
C	0.1	0.698	0.692	0.688	0.675
		0.1	0.2	0.3	0.4
		Alpha			

Figure 24: Working of GridSearchCV

6.7.2 <u>RandomizedSearchCV</u>

RandomizedSearchCV solves the drawbacks of GridSearchCV, as it goes through only a fixed number of hyperparameter settings. It moves within the grid in random fashion to find the best set hyperparameters. This approach reduces unnecessary computation.



Hyperparameter Tuning

```
In [243]: random_forest_parameters =  {'n_estimators' : list(range(2,200,25)),
                                     'criterion' : ['gini','entropy'],
                                     'max_depth': [1,2,3,5,13,15, None],
                                     'min_samples_split' : list(range(2,10,4))}

In [246]: random_search_random_forest = RandomizedSearchCV(estimator=random_forest,
                                                           param_distributions=random_forest_paramete
                                                           cv=10, scoring='accuracy',n_iter = 100)

In [247]: random_search_random_forest.fit(train_set_x,train_set_y)
```

```
RandomizedSearchCV(cv=10, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_job...
                                                    oob_score=False,
                                                    random_state=42, verbose=0,
```

Figure 25 Hyperparameter Tuning

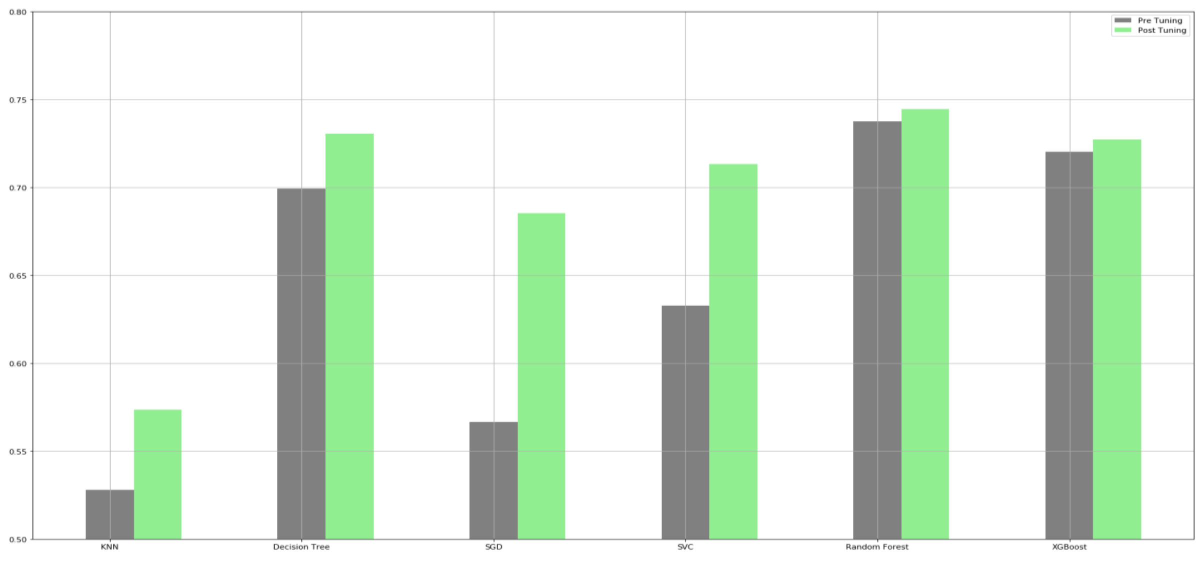


Figure 26 Plot to show accuracy improvement post hyperparameter tuning

- Improvement in accuracy of various models after tuning their hyperparameters.

Chapter 7: Model Evaluation Using Performance Measures

7.1 K-Fold Cross Validation

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods. It divides the data into 'K' parts and trains 'K-1' parts and tests the model on onw part repeating the process K times.

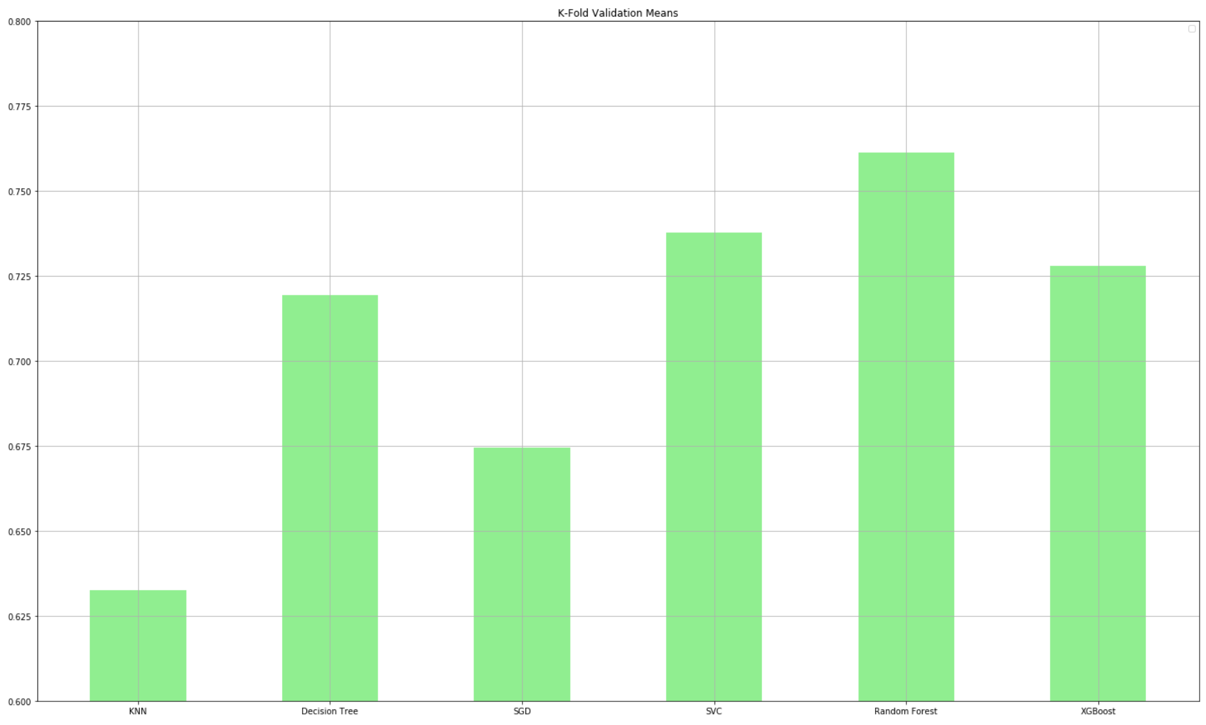
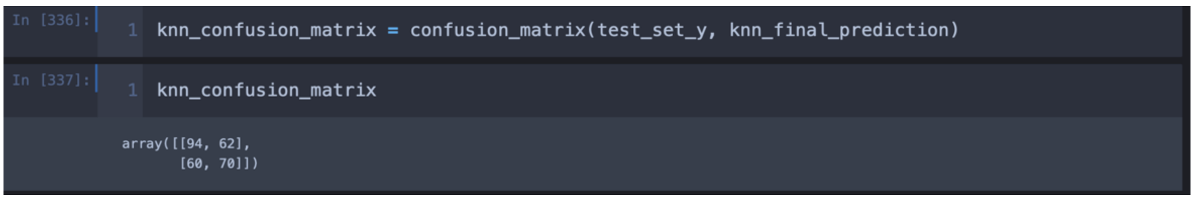


Figure 27 K-Fold Validation scores of all models

It is a very reliable performance measure based on the observations from the above plot we will try restricting further performance analysis to 3-4 models only.

7.2 Confusion Matrix

The sklearn.metrics provides a function that returns a confusion matrix.



```
In [336]:  knn_confusion_matrix = confusion_matrix(test_set_y, knn_final_prediction)

In [337]:  knn_confusion_matrix

           array([[94, 62],
                  [60, 70]])
```

Figure 28 Code for Confusion Matrix

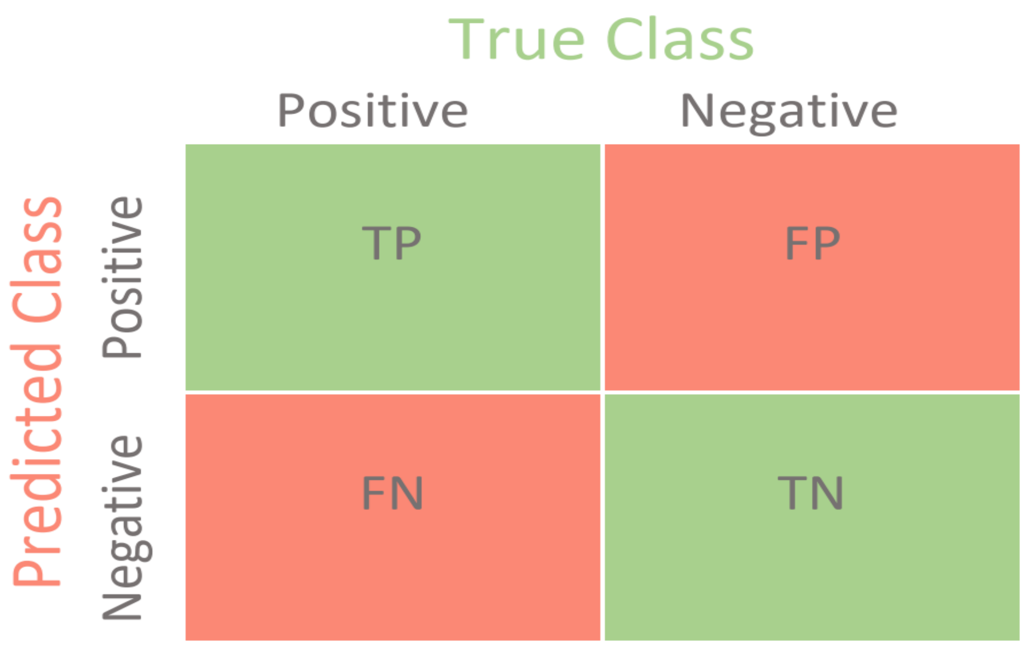
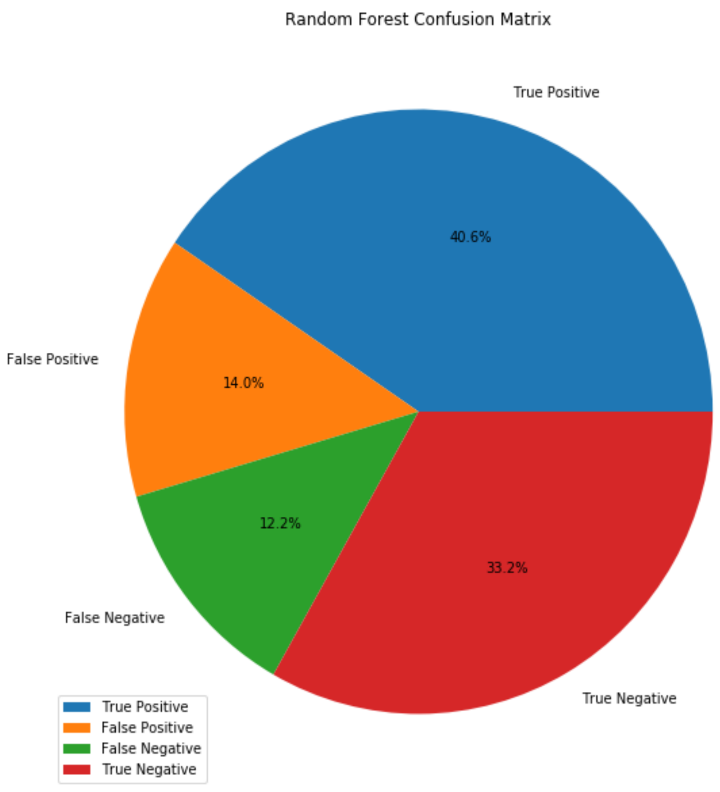
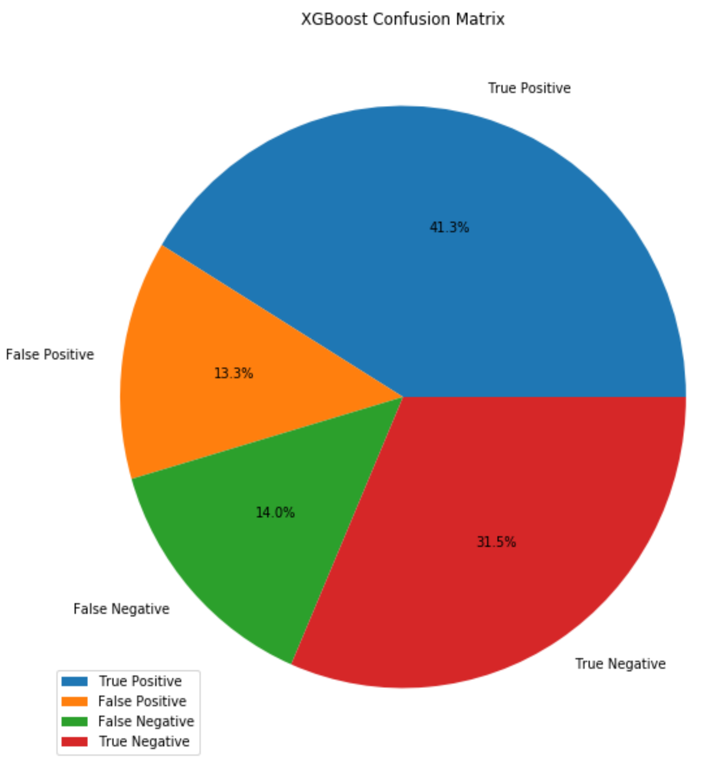
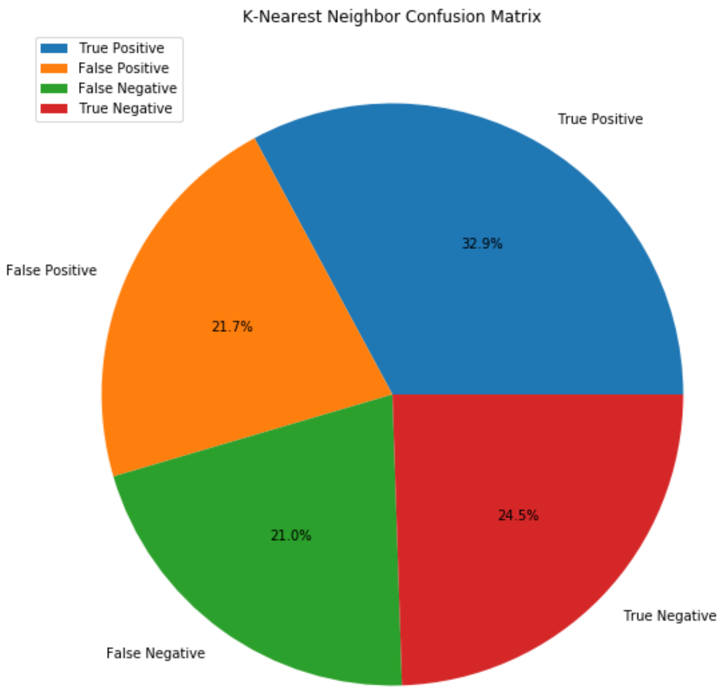


Figure 29 Confusion Matrix Diagram

Confusion matrix are very useful in analysing performance measures such as Precision, Recall and Accuracy.




XGBoost Confusion Matrix
True Positive
41.3%
False Positive
13.3%
14.0%
False Negative
31.5%
True Negative
True Positive
False Positive
False Negative
True Negative


K-Nearest Neighbor Confusion Matrix
True Positive
False Positive
False Negative
True Negative
True Positive
32.9%
False Positive
21.7%
24.5%
21.0%
True Negative
False Negative

From these pie charts we can observe the ratio of True Positive, False Positive, False Negative and True Negative for different models.

To appreciate what these pie charts indicate we will have to first study precision and recall.

7.3 Beyond Accuracy: Precision and Recall

Precision and recall become particularly important in cases where there is large imbalance in target values in the dataset. For instance while designing a Machine Learning algorithm that predict shoplifters from a large customer database that contains data records of 1 million customers of which are only 1 thousand are actually shoplifters. If. A model predicts all 1 million to be non-shoplifters. It will give us an accuracy of 99.9%, it might look good, but it is very obvious as that this model doesn't serve the purpose of identifying shoplifters.

Precision = True Positive/(True Positive + False Positive)

Recall = True Positive/(True Positive + False Negative)

An ideal classifier has both Precision and Recall equal to 1.

The metric our intuition tells us we should maximize is known in statistics as **recall**, or the ability of a model to find all the relevant cases within a dataset. The precise definition of recall is the number of true positives divided by the number of true positives plus the number of false negatives. True positives are data point classified as positive by the model that are positive (meaning they are correct), and false negatives are data points the model identifies as negative that actually are positive (incorrect). The importance of precision and recall varies with problem. Some cases we might need high precision and in some high recall is needed.

Let us try to look at what is important for our project. From our personal experience we can say that say that we rather hear few bad songs rather than let good ones get away from us…. By that logic Recall is our answer.

Or

If hearing a song against your test messes with your head, then go for Precision. As high precision means low number of False Positives.

7.4 F1 Score

F1 score is used to measure balance between precision and recall. As precision and recall are inversely proportional.

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

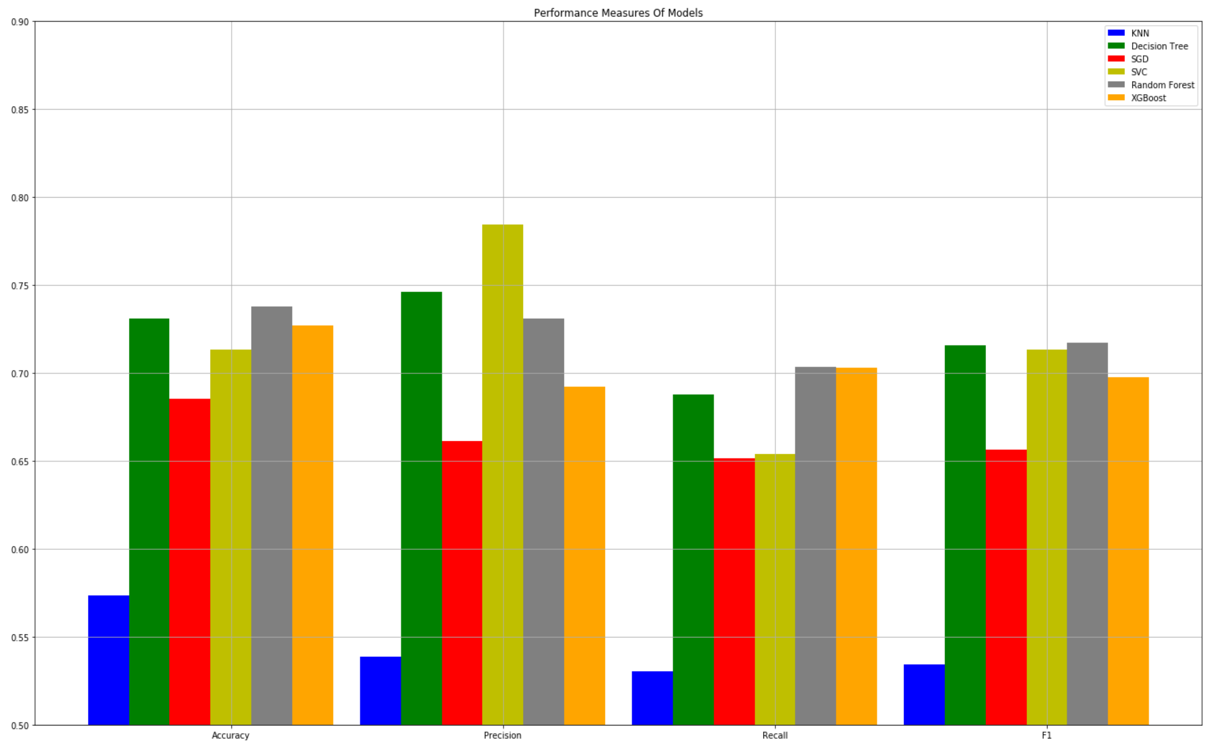


Figure 30 Performance of All Models

7.5 Precision-Recall Trade-Off

Like we mentioned earlier precision and recall are inversely proportional. Let us look at a few plots to get a clearer picture of the trade-off between the two.

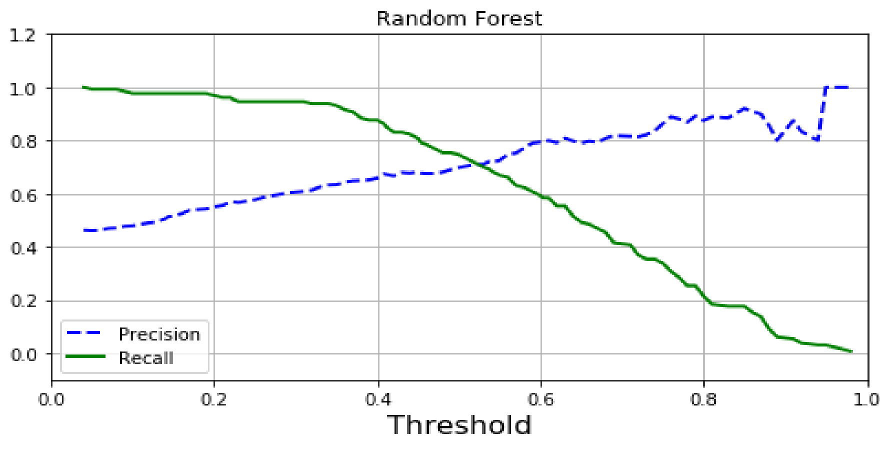


Figure 31 Random Forest Precision-Recall Trade-Off

From the graph we can observe that precision and recall are equal when threshold is slightly greater than 0.5 and movement of threshold to the left increases recall and on the right increases precision.

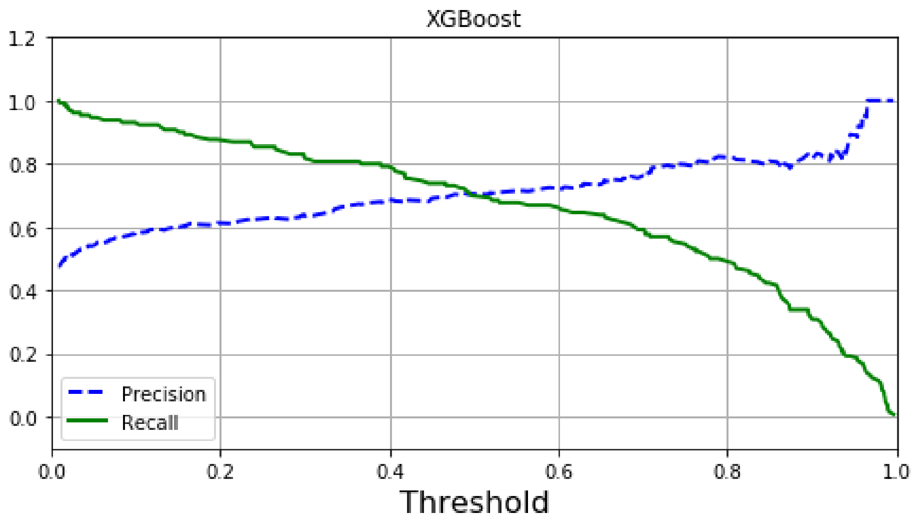


Figure 32 XG-Boost Precision Recall Trade-Off

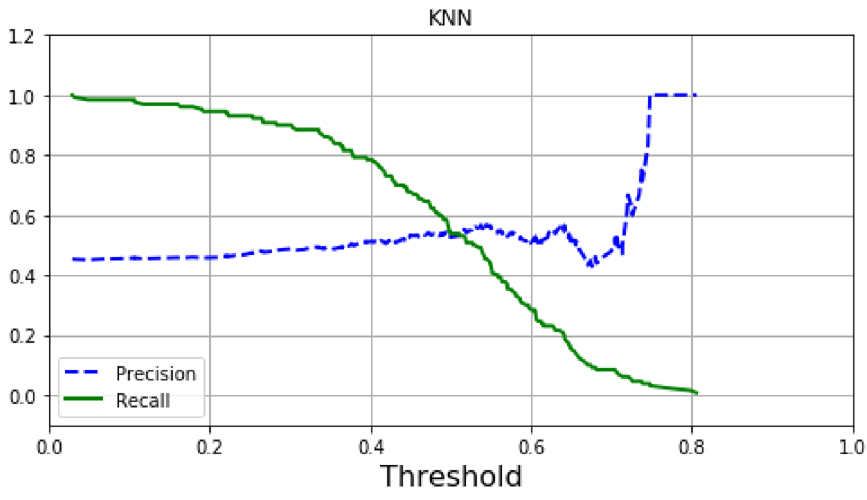


Figure 33 K-Nearest Neighbor Precision-Recall Trade-Off

- From these plots it can be observed that for Random Forest and XGBoost we can move threshold to the left and get 80% recall and 70%precision whereas for KNN for 80% recall the precision fall below 50%.
- Between Random Forest and XG-Boost Random Forest appears to be better for higher precision values.

In Machine Learning, performance measurement is an essential task. So, when it comes to a classification problem, we can count on an AUC - ROC Curve. When we need to check or visualize the performance of the multi - class classification problem, we use AUC (**Area Under the Curve**) ROC (**Receiver Operating Characteristics**) curve. It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (**Area Under the Receiver Operating Characteristics**)

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represent degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between patients with disease and no disease.
The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.

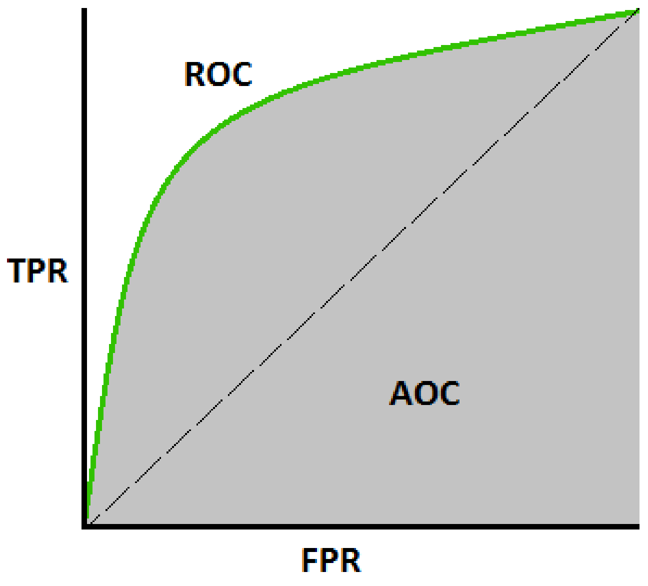


Figure 34 ROC Curve

7.5.1 <u>Defining Terms Used in AUC and ROC Curve</u>

$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

$$\text{FPR} = 1 - \text{Specificity}$$

$$= \frac{FP}{TN + FP}$$

An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact, it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity whatsoever.

7.5.2 <u>Relation Between Sensitivity, Specificity, FPR and Threshold</u>

Sensitivity and Specificity are inversely proportional to each other. So, when we increase Sensitivity, Specificity decreases and vice versa.

Sensitivity ⬆, Specificity ⬇ and Sensitivity ⬇, Specificity ⬆

When we decrease the threshold, we get more positive values thus it increases the sensitivity and decreasing the specificity.

Similarly, when we increase the threshold, we get more negative values thus we get higher specificity and lower sensitivity.

As we know FPR is 1 - specificity. So, when we increase TPR, FPR also increases and vice versa.

TPR ⬆, FPR ⬆ and TPR ⬇, FPR ⬇

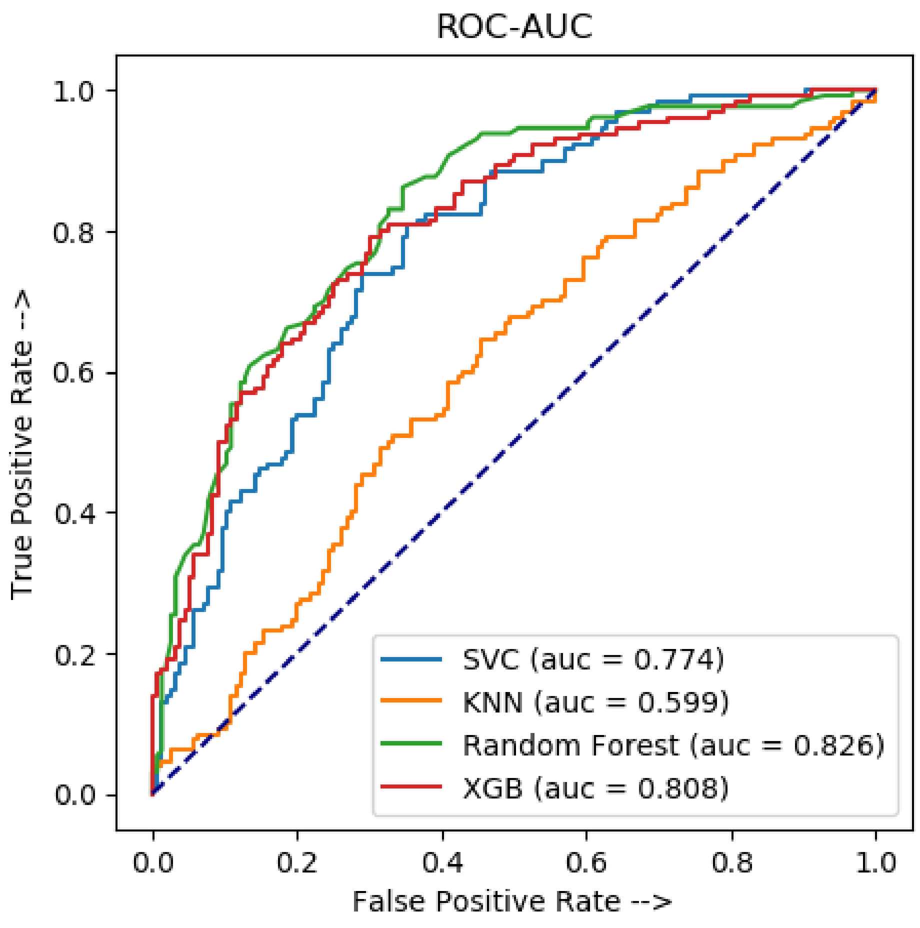
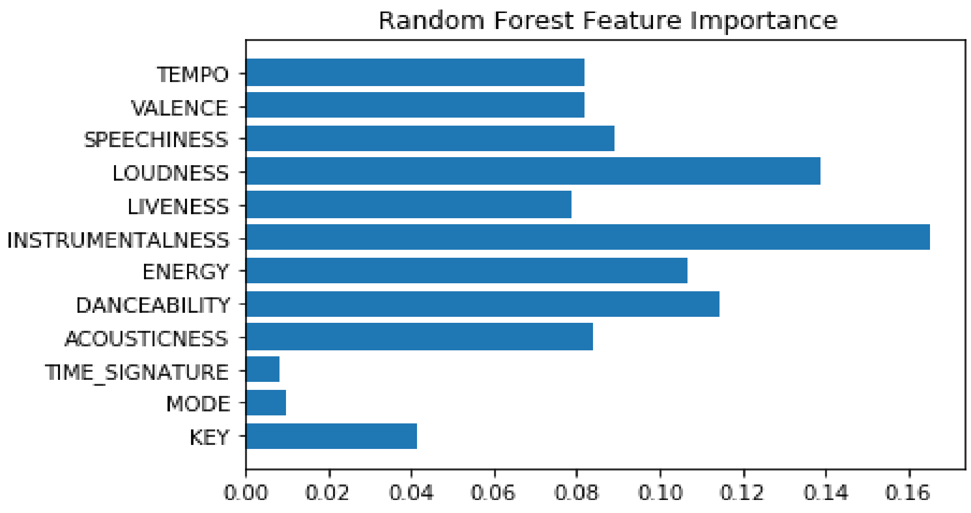


Figure 35 ROC-AUC of Various Models

7.6 Feature Importance



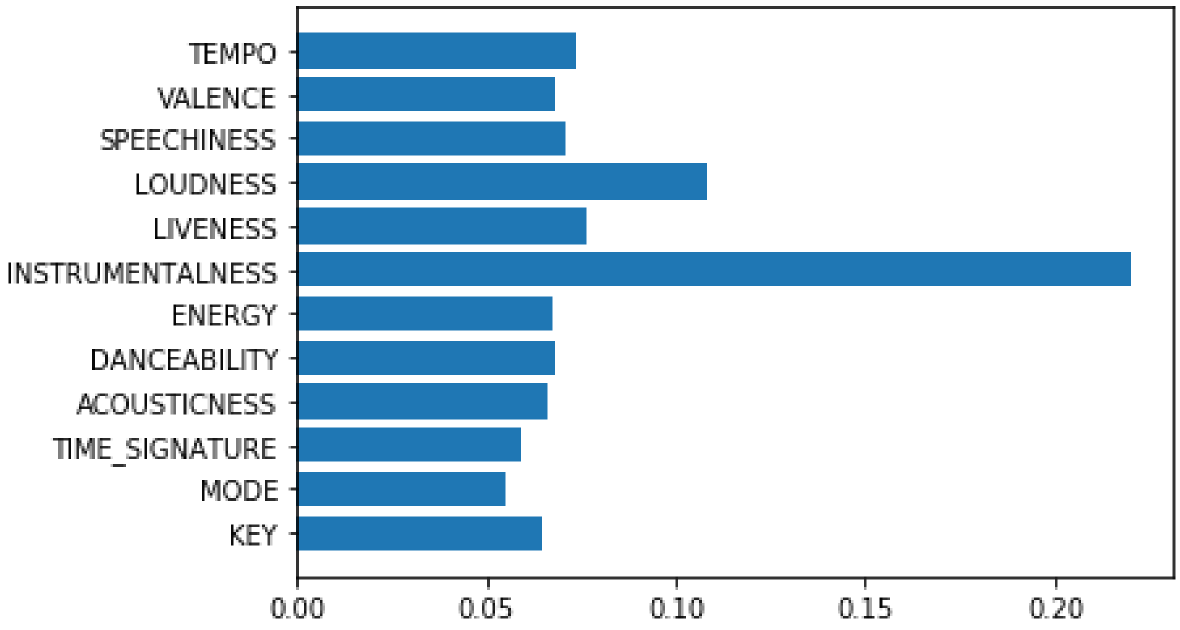
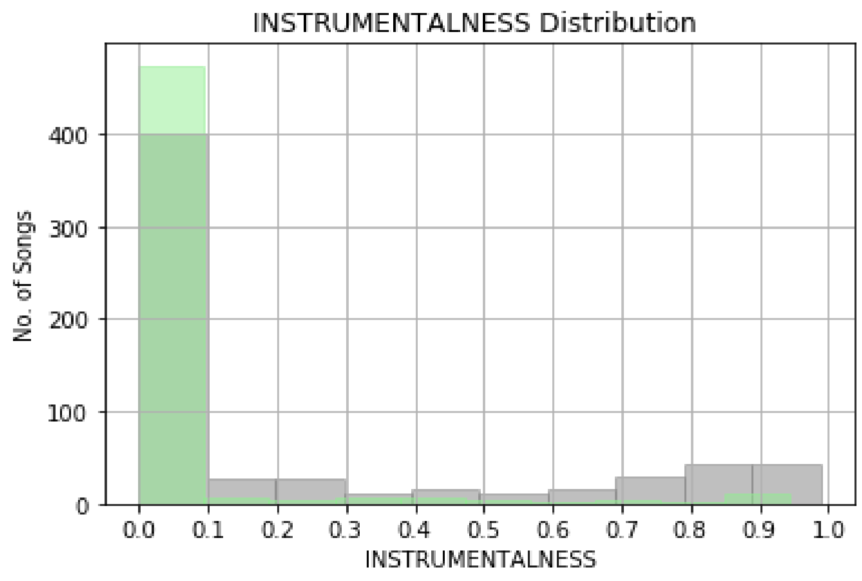


Figure 36 XG-Boost Feature Importance

From the above plots it is evident that instrumentalness is an important feature for Random Forest and XG-Boost, which are the best performing models.



- Looking at the above plot during EDA we did not see any strong correlation of Instrumentalness with target values.
- Like we predicted loudness, danceability also turned out to be important.
- Feature Scaling is whole another topic beyond the scope of this project to explore and get even better results.

Chapter 8: Conclusion

1. Despite not getting any discernible insights from the EDA and given the subjective nature of the problem ROC-AUC of 82% is a satisfactory result.
2. We intentionally did not take the artists attribute into account as it would make the task less complex and increase bias.
3. We analysed why using RandomSearhCV over GridSearchCV is a better choice for hyperparameter tuning as both give almost identical results but the computation cost of RandomSearchCV is 0.00001% of GridSearhCV.
4. We studied how accuracy can be a misleading performance measure.
5. The relationship and balance between Precision and Recall, the different cases in which they become important.
6. In case of other data sets such as House Pricing Dataset and Image Classification datasets we can look at the the data attributes for a row and in most cases we can tell the target values for that row but in this case we can’t do given the attributes of a song and therefore the results are more than satisfactory.
7. We could not try combining attributes as it requires knowledge about Feature Scaling and better knowledge about these song attributes.
8. Lastly in this project we treated ML models as black boxes, feeding data, and getting results. Ideally, we would like to understand the maths that is behind all these algorithms, so we have sound knowledge to make changes when necessary and totally depend on pre-defined functions for the job.

Chapter 9: References

- https://opendatascience.com/a-machine-learning-deep-dive-into-my-spotify-data/
- https://www.kaggle.com/pmarcelino/data-analysis-and-feature-extraction-with-python
- https://www.kaggle.com/lowkimhoe/prediction-model-on-spotify-classification
- https://machinelearningmastery.com/k-fold-cross-validation/
- https://towardsdatascience.com/3-things-you-need-to-know-before-you-train-test-split-869dfabb7e50
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/
- https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
- https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- https://scikit-learn.org/stable/modules/sgd.html
- https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=278443377086&utm_targetid=aud-299261629574:dsa-429603003980&utm_loc_interest_ms=&utm_loc_physical_ms=9301805&gclid=Cj0KCQiAh4j-BRCsARIsAGeV12DvJWJxvC31RxxP2BgdI9B87TdFyILKTtSMbOS9sB8HNCd0b3N8BtsaAkS6EALw_wcB
- https://scikit-learn.org/stable/modules/neighbors.html#classification
- https://medium.com/swlh/decision-tree-classification-de64fc4d5aac
- https://scikit-learn.org/stable/modules/tree.html#
- https://scikit-learn.org/stable/modules/tree.html#classification
- https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- https://scikit-learn.org/stable/modules/ensemble.html#forest
- https://xgboost.readthedocs.io/en/latest/tutorials/rf.html
- https://towardsdatascience.com/a-beginners-guide-to-xgboost-87f5d4c30ed7
- Hands on Machine Learning by Aurélien Géron