INTERNSHIP REPORT

ON

MUSIC CLASSIFICATION AND RECOMMENDATION

Submitted in complete fulfillment for

Machine Learning & Artificial Intelligence Internship

SUBMITTED

BY

RAHUL KOLAY
19BEC0169
Electronics and Communication, 5th Semester
Vellore Institute of Technology,Vellore

ANUSHKAA AWASTHI
209309057
Data Science and Engineering, 3rd Semester
Manipal University, Jaipur

S SHRADDHA
4CB19IS048
Information science and Engineering,5th Semester
Canara Engineering college,Mangalore

UNDER THE GUIDANCE OF

Mr. Shashank,

Technical Lead,

QuantMasters

Quant Masters Pvt. Ltd,

#812, 6th cross 3rd main, Rajajinagar, Bengaluru - 560021

Table of Contents

ABSTRACT

The Widespread usage of internet has brought about significant changes in the music industry as well as causing all kinds of change. Example of these developments including widespread use of online music listening and sales platform, control of music copyright, classification of music genre,and recommendations. Today with the advancement of music broadcast platforms, people can listen to music at any time and anywhere in and can be reached millions of songs through various music listening platforms like Spotify,last.fm and many more.

Music genre prediction is one of the topics that digital music processing is interested in. In this study, acoustic features of music have been extracted by using digital signal processing techniques and then music genre classification and music recommendations have been made by using machine learning methods. In the study, GTZAN database has been used and the highest success was obtained with the SVM algorithm.

INTRODUCTION

1.1 What is Music Classification?

A music genre is a conventional category that identifies some pieces of music as belonging to a shared tradition or set of conventions.It is to be distinguished from *musical form* and musical style, although in practice these terms are sometimes used interchangeably.

Music can be divided into genres in varying ways, such as popular music and art music, or religious music and secular music. The artistic nature of music means that these classifications are often subjective and controversial, and some genres may overlap.

A *subgenre* is a subordinate within a genre. In music terms, it is a subcategory of a musical genre that adopts its basic characteristics, but also has its own set of characteristics that clearly distinguish and set it apart within the genre. A subgenre is also often being referred to as a style of the genre. The proliferation of popular music in the 20th century has led to over 1,200 definable subgenres of music.

A musical composition may be situated in the intersection of two or more genres, sharing characteristics of every parent genre and therefore belong to each genre of these at the same time,such subgenres are known as *fusion genres*. Examples of fusion genres include jazz fusion, which is a fusion of jazz and rock music, and country rock which is a fusion of country music and rock music.

A *microgenre* is a niche genre,as well as a subcategory within major genres or their subgenres.

1.2 What is Machine Learning?

Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, uncovering key insights within data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth metrics. As big data continues to expand and grow, the market demand for data scientists will increase, requiring them to assist in the identification of the most relevant business questions and subsequently the data to answer them.

Machine Learning Methods:

(a) Supervised machine learning

Supervised learning, also known as supervised machine learning, is defined by its use of labeled datasets to train algorithms that to classify data or predict outcomes accurately. As input data is fed into the model, it adjusts its weights until the model has been fitted appropriately. This occurs as part of the cross validation process to ensure that the model avoids overfitting or underfitting. Supervised learning helps organizations solve for a variety of real-world problems at scale, such as classifying spam in a separate folder from your inbox. Some methods used in supervised learning include neural networks, naïve bayes, linear regression, logistic regression, random forest, support vector machine (SVM), and more.

Unsupervised machine learning

Unsupervised learning, also known as unsupervised machine learning, uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, cross-selling strategies, customer segmentation, image and pattern recognition. It's also used to reduce the number of features in a model through the process of dimensionality reduction; principal component analysis (PCA) and singular value decomposition (SVD) are two common approaches for this. Other algorithms used in unsupervised learning include neural networks, k-means clustering, probabilistic clustering methods, and more.

Semi-supervised learning

Semi-supervised learning offers a happy medium between supervised and unsupervised learning. During training, it uses a smaller labeled data set to guide classification and feature extraction from a larger, unlabeled data set. Semi-supervised learning can solve the problem of having not enough labeled data (or not being able to afford to label enough data) to train a supervised learning algorithm.

RELATED WORK

Machine learning techniques have been used for music genre classification for decides now. In 2002, G. Tzanetakis and P. Cook [8] used both the mixture of Gaussians model and k-nearest neighbors along with three sets of carefully hand-extracted features representing timbral texture, rhythmic content and pitch content. They achieved 61% accuracy.

As a benchmark, human accuracy averages around 70% for this kind of genre classification work [4]. Tzanetakis and Cook used MFCCs, a close cousin of mel-spectrograms, and essentially all work has followed in their footsteps in transforming their data in this manner. In the following years, methods such as support vector machines were also applied to this task, such as in 2003 when C. Xu et al. [9] used multiple layers of SVMs to achieve over 90% accuracy on a dataset containing only four genres.In the past 5-10 years, however, convolutional neural networks have shown to be incredibly accurate music genre classifiers [8] [2] [6], with excellent results reflecting both the complexity provided by having multiple layers and the ability of convolutional layers to effectively identify patterns within images (which is essentially what mel-spectrograms and MFCCs are). These results have far exceeded human capacity for genre classification, with our research finding that current state-of-the-art models perform with an accuracy of around 91% [6] when using the full 30s track length. Many of the papers which implemented CNNs compared their models to other ML techniques, including k-NN, mixture of Gaussians, and SVMs, and CNNs performed favorably in all cases. Therefore we decided to focus our efforts on implementing a high-accuracy CNN, with other models used as a baseline.

This study aims to classify and recommend songs using acoustic features, extracted by digital signal processing methods. Study has been conducted over two steps; determining how features will be used in recommendation are obtained and developing a service that recommends songs to user requests.

Automatic genre classification is a difficult and problematic task that nevertheless has important value in terms of both pure research and commercial application. Continuing research in automatic genre classification has much to offer,as does parallel research involving other aspects of musical similarity.The suggestions would be:

- Each recording should be permitted to have more than one genre label, and labels should be waited.
- Information from high level,low level and cultural features should be combined.

Most importantly all areas of research from a concerted effort to develop carefully annotated music datasets that include varied metadata such as genre, mood, groove, composer,performer, lyrics,meter,chord progressions, instrument presents etc. According to the results summarised,SVM achieved better classification results than any other methods.

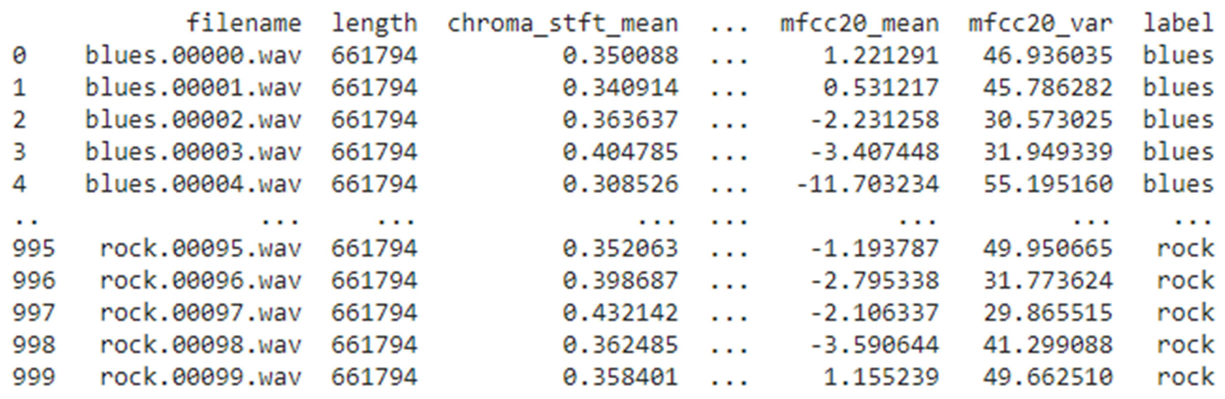
SYSTEM ARCHITECTURE

The data set consist of one thousand music files with hundred files of each type. The different type of music files are:-

1. Blues
2. Classical
3. Country
4. HipHop
5. Disco
6. Jazz
7. Metal

8. Pop
9. Reggae
10. Rock

The datasets consist of each file name and its length,chroma shift mean,chroma shift variance, rms mean,rms variance,spectral centroid mean,spectral centroid variance.spectral bandwidth mean, spectral bandwidth variance, roll-of mean, roll-of variance, zero crossing rate mean,zero crossing rate variance,harmony mean,harmony variance, percepter mean, percepter variance,tempo,**Mel-frequency cepstral coefficients**(mfcc) mfcc1 mean, mfcc1 variance, mfcc2 mean, mfcc2 variance, mfcc3 mean, mfcc3 variance, mfcc4 mean, mfcc4 variance, mfcc5 mean, mfcc5 variance, mfcc6 mean, mfcc6 variance, mfcc7 mean, mfcc7 variance, mfcc8 mean, mfcc8 variance, mfcc9 mean, mfcc9 variance, mfcc10 mean, mfcc10 variance, mfcc11 mean, mfcc11 variance, mfcc12 mean, mfcc12 variance, mfcc13 mean, mfcc13 variance, mfcc14 mean, mfcc14 variance, mfcc15 mean, mfcc15 variance, mfcc16 mean, mfcc16 variance, mfcc17 mean, mfcc17 variance, mfcc18 mean, mfcc18 variance, mfcc19 mean, mfcc19 variance, mfcc20 mean, mfcc20 variance, label.



```
            filename  length  chroma_stft_mean  ...  mfcc20_mean  mfcc20_var  label
0    blues.00000.wav  661794          0.350088  ...     1.221291   46.936035  blues
1    blues.00001.wav  661794          0.340914  ...     0.531217   45.786282  blues
2    blues.00002.wav  661794          0.363637  ...    -2.231258   30.573025  blues
3    blues.00003.wav  661794          0.404785  ...    -3.407448   31.949339  blues
4    blues.00004.wav  661794          0.308526  ...   -11.703234   55.195160  blues
..               ...     ...               ...  ...          ...         ...    ...
995   rock.00095.wav  661794          0.352063  ...    -1.193787   49.950665   rock
996   rock.00096.wav  661794          0.398687  ...    -2.795338   31.773624   rock
997   rock.00097.wav  661794          0.432142  ...    -2.106337   29.865515   rock
998   rock.00098.wav  661794          0.362485  ...    -3.590644   41.299088   rock
999   rock.00099.wav  661794          0.358401  ...     1.155239   49.662510   rock
```

Software used: Google Colab Machine Learning with Python.

Packages used:

1. import librosa
2. import numpy
3. import pandas
4. import matplotlib

5. from sklearn.model_selection import train_test_split
6. from sklearn.naive_bayes import GaussianNB
7. from sklearn.linear_model import SGDClassifier, LogisticRegression
8. from sklearn.neighbors import KNeighborsClassifier
9. from sklearn.tree import DecisionTreeClassifier
10. from sklearn.ensemble import RandomForestClassifier
11. from sklearn.svm import SVC
12. from sklearn.neural_network import MLPClassifier
13. from xgboost import XGBClassifier, XGBRFClassifier
14. from xgboost import plot_tree, plot_importance
15. from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score, roc_curve, f1_score
16. from sklearn import preprocessing
17. from sklearn.feature_selection import RFE
18. from sklearn.linear_model import LinearRegression
19. from sklearn.preprocessing import StandardScaler
20. import IPython.display
21. from sklearn.metrics.pairwise import cosine_similarity
22. from sklearn import preprocessing

READING MUSIC FILE:

```
# Importing 1 file

import librosa

y, sr = librosa.load(/'path of music file')
```

=> Read a music file.

Read all features of a music file using librosa libraries and store them in a csv file with filename and its type.

READING DATA:

The data set is read using **pd.read_csv('path of the file')**.

The csv file is in the google drive of the programmer and the colab is mounted with the drive.

Apart from the filename all features are independent features. The '**label**' is the dependent feature . The music files are trained and tested, and they are classified to which type of music they belong by providing all the independent features.

Classification Algorithms used:

1. K Neighbour Classification:
2. Support Vector Machine Classification:
3. Decision Tree Classification
4. Random Forest Classification
5. Naive Bayes Classification
6. XGBoost Classification
7. XGBRF Classification
8. MLP Classification

Divide data to independent and dependent features:

All the audio features except the label are as independent features in 'x'.

All the labels are dependent features in 'y'.

Scaling of dependent features:

```
cols = x.columns

min_max_scaler = preprocessing.MinMaxScaler()

np_scaled = min_max_scaler.fit_transform(x)

# new data frame with the new scaled data.

x = pd.DataFrame(np_scaled, columns = cols)
```

This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

All are numeric datas.

Divide the datas to train and test set:-

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
```

70% is used as training data and is fitted into different Classification models

30% is used as test data to predict the dependent feature.

1. **K Neighbors Classification:**

```
from sklearn.neighbors import KNeighborsClassifier

k_nearest_classifier = KNeighborsClassifier(n_neighbors=10,p=2)
```

```
k_nearest_classifier.fit(x_train,y_train)
```

```
=> KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',metric_params=None, n_jobs=None, n_neighbors=10,
p=2,weights='uniform')
```

Mostly we need to alter the ‘p’ and ‘n_neighbors’ to determine accuracy and precision of each K_neighbor model. we will also calculate the F1_score and area of the ROC Curve.

2. **Support Vector Machine Classifier:**

```
from sklearn.svm import SVC

svc_classifier = SVC(C=2,kernel='rbf')

svc_classifier.fit(x_train,y_train)
```

```
=> SVC(C=2, 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)
```

Mostly we need to alter the ‘C’ , ‘kernel’ and ‘gamma’ to determine accuracy and precision of each SVC model. we will also calculate the F1_score and area of the ROC Curve.

3. **Decision Tree Classifier:**

```
dtc= DecisionTreeClassifier(criterion='entropy',max_features=None)

dtc.fit(x_train,y_train)
```

```
=>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=None, splitter='best')
```

Mostly we need to alter the 'criterion' to determine accuracy and precision of each SVC model. we will also calculate the F1_score and area of the ROC Curve.

4. **Random Forest Classification:**

```
from sklearn.ensemble import RandomForestClassifier

rforest = RandomForestClassifier(n_estimators=4000, max_depth=40, random_state=0)

rforest.fit(x_train,y_train)

=> RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,criterion='gini', max_depth=40,
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=4000,n_jobs=None,
oob_score=False, random_state=0, verbose=0,warm_start=False)
```

5. **MLPClassifier:**

```
clf = MLPClassifier(random_state=1, max_iter=300).fit(x_train, y_train)

=> self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto',
beta_1=0.9,beta_2=0.999, early_stopping=False, epsilon=1e-
08,hidden_layer_sizes=(5000, 10), learning_rate='constant',learning_rate_init=0.001,
max_fun=15000, max_iter=200,momentum=0.9, n_iter_no_change=10,
nesterovs_momentum=True,power_t=0.5, random_state=1, shuffle=True,
solver='lbfgs',tol=0.0001, validation_fraction=0.1, verbose=False,warm_start=False)
```

6. **Naive Bayes Algorithm:**

```
gnb = GaussianNB()

gnb.fit(x_train, y_train)
```

7. **XG Boost Classifier:**

```
xgb = XGBClassifier(n_estimators=3500, learning_rate=0.05,base_score=1)

xgb.fit(x_train,y_train)

=> XGBClassifier(base_score=1, booster='gbtree',
colsample_bylevel=1,colsample_bynode=1, colsample_bytree=1,
gamma=0,learning_rate=0.05, max_delta_step=0, max_depth=3,min_child_weight=1,
missing=None, n_estimators=3500, n_jobs=1,nthread=None, objective='multi:softprob',
random_state=0,reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
seed=None,silent=None, subsample=1, verbosity=1)
```

8. XGBRF Classifier:

```
xgbrf = XGBRFClassifier(objective= 'multi:softmax')
```

```
xgbrf.fit(x_train,y_train)

=>  XGBRFClassifier(base_score=0.5, colsample_bylevel=1,
colsample_bynode=0.8,colsample_bytree=1, gamma=0, learning_rate=1,
max_delta_step=0,max_depth=3, min_child_weight=1, missing=None,
n_estimators=100,n_jobs=1, nthread=None, objective='multi:softprob',random_state=0,
reg_alpha=0, reg_lambda=1, scale_pos_weight=1,seed=None, silent=None,
subsample=0.8, verbosity=1)
```

ACCURACY , PRECISION, F1-SCORE,CONFUSION MATRIX AND ROC SCORE:-

obj= Object of the classifier

```
y_pred = obj.predict(x_test)

print("Accuracy: ",accuracy_score(y_test,y_pred)*100)

c_m = confusion_matrix(y_test,y_pred)

print(c_m)

from sklearn.metrics import precision_score

prec = precision_score(y_test,y_pred,average='macro')

print(prec*100)

from sklearn.metrics import f1_score

f1 = f1_score(y_test,y_pred,average='macro')
```

```
print(f1*100)

#ROC Curve Analysis

predicted_probabilities = obj.predict_proba(x_test)

from sklearn.metrics import roc_curve

fpr,tpr,threshold = roc_curve(y_test,predicted_probabilities[:,1],pos_label=1)

plt.style.use('seaborn')

plt.plot(fpr,tpr,linestyle='--')

plt.show()

#auc_score

from sklearn.metrics import roc_auc_score

roc_auc_score(y_test,predicted_probabilities,multi_class='ovo')
```

Therefore, using this code , these results are calculated for each classifier algorithm and the best model is chosen.

GRID SEARCH:-

```
from sklearn.model_selection import GridSearchCV

parameters_grid =
[{'parameter1'':[1000,500,1500,2000,2500,3000,4000,3500,5000],'parameter2':[0.05],'par
ameter3'':[0.05,0.1,0.5,1,10,100]}]
```

```
grid_search_results = GridSearchCV(estimator = obj,param_grid =
parameters_grid,scoring = 'accuracy',n_jobs = -1,cv = 10)

grid_search_results = grid_search_results.fit(x_train,y_train)

grid_search_results.best_score_

grid_search_results.best_params_
```

Therefore, it gives the best parameters values of a classifier that gives the best accuracy.

RECOMMENDATION ALGORITHM:

```
import IPython.display as ipd

from sklearn.metrics.pairwise import cosine_similarity

from sklearn import preprocessing



# Read data

data = pd.read_csv('/content/drive/MyDrive/features_3_sec.csv', index_col='filename')
```

=> Selecting a audio file from the csv file using its filename.

```
# Extract labels
```

```
labels = data[['label']]


# Drop labels from original dataframe

data = data.drop(columns=['length','label'])

data.head()


# Scale the data

data_scaled=preprocessing.scale(data)

print('Scaled data type:', type(data_scaled))
```

=> Scaling the data.

```
# Cosine similarity

similarity = cosine_similarity(data_scaled)

print("Similarity shape:", similarity.shape)


# Convert into a dataframe and then set the row index and column names as labels

sim_df_labels = pd.DataFrame(similarity)

sim_df_names = sim_df_labels.set_index(labels.index)
```

sim_df_names.columns = labels.index

sim_df_names.head()

=> Similarity using its shape.

```
def find_similar_songs(name):

    # Find songs most similar to another song

    series = sim_df_names[name].sort_values(ascending =
False)


    # Remove cosine similarity == 1 (songs will always have
the best match with themselves)

    series = series.drop(name)


    # Display the 5 top matches

    print("\n*******\nSimilar songs to ", name)

    print(series.head())
```

Therefore, this algorithm recommends five similar songs from the csv file with respect to certain parameters.

RESULTS

*In each case the given parameters were considered after applying grid search or trial and error to each model.

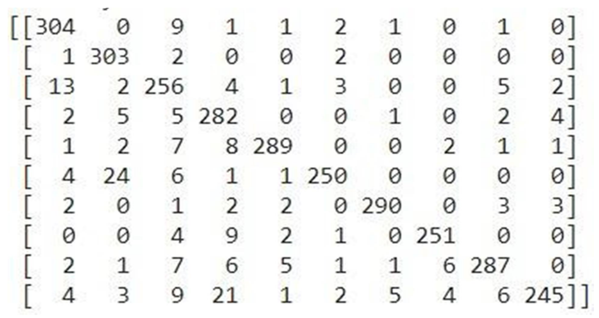
The following alogorithms were considered for the given data and these were the results obtained:

4.1: KNeighborsClassifier

KNeighborsClassifier(*n_neighbors=2, p=1*)

n_neighbors(=2) indicates the number of neighbours to use by the classifier for kneighbors queries. All other values are taken as default.

Confusion Matrix:



```
[[304   0   9   1   1   2   1   0   1   0]
 [  1 303   2   0   0   2   0   0   0   0]
 [ 13   2 256   4   1   3   0   0   5   2]
 [  2   5   5 282   0   0   1   0   2   4]
 [  1   2   7   8 289   0   0   2   1   1]
 [  4  24   6   1   1 250   0   0   0   0]
 [  2   0   1   2   2   0 290   0   3   3]
 [  0   0   4   9   2   1   0 251   0   0]
 [  2   1   7   6   5   1   1   6 287   0]
 [  4   3   9  21   1   2   5   4   6 245]]
```

This method gave an

Accuracy of 91.991991991992

Precision of 92.29105022723694

F1 Score of 91.97161448784505

Area under Curve Percentage of 99.27071784913669

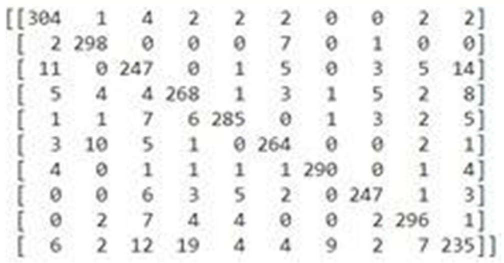
This algorithm gives us the best results. As we can see most of the values lie along the diagonal of the confusion matrix and it gives us the best accuracy as well. Even though XBG classifier shows a similar roc_auc value, the accuracy, precision and f1 score offered by KNeighborsClassifier algorithm is much better.

4.2: SVC

SVC(C=100,kernel='rbf', gamma=1)

It is the regularization parameter. The strength of the regularization is inversely proportional to C(=100). All other parameters are set to default.

Confusion Matrix:



```
[[304   1   4   2   2   2   0   0   2   2]
 [  2 298   0   0   0   7   0   1   0   0]
 [ 11   0 247   0   1   5   0   3   5  14]
 [  5   4   4 268   1   3   1   5   2   8]
 [  1   1   7   6 285   0   1   3   2   5]
 [  3  10   5   1   0 264   0   0   2   1]
 [  4   0   1   1   1   1 290   0   1   4]
 [  0   0   6   3   5   2   0 247   1   3]
 [  0   2   7   4   4   0   0   2 296   1]
 [  6   2  12  19   4   4   9   2   7 235]]
```

This method gave an

Accuracy of 91.22455789122455

Precision of 91.17954027880485

F1 Score of 91.14004562493034

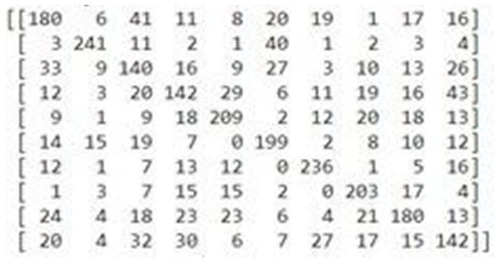
Area under Curve Percentage of 94.34253398008761

4.3: DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy',max_features=15)

Entropy is the function to measure the quality of a split based on the criteria for the information gain. It considers 15 features at a time while making the split.

Confusion Matrix:



```
[[180   6  41  11   8  20  19   1  17  16]
 [  3 241  11   2   1  40   1   2   3   4]
 [ 33   9 140  16   9  27   3  10  13  26]
 [ 12   3  20 142  29   6  11  19  16  43]
 [  9   1   9  18 209   2  12  20  18  13]
 [ 14  15  19   7   0 199   2   8  10  12]
 [ 12   1   7  13  12   0 236   1   5  16]
 [  1   3   7  15  15   2   0 203  17   4]
 [ 24   4  18  23  23   6   4  21 180  13]
 [ 20   4  32  30   6   7  27  17  15 142]]
```

This method gave an

Accuracy of 62.46246246246246

Precision of 62.36169432075411

F1 Score of 62.3965072650474

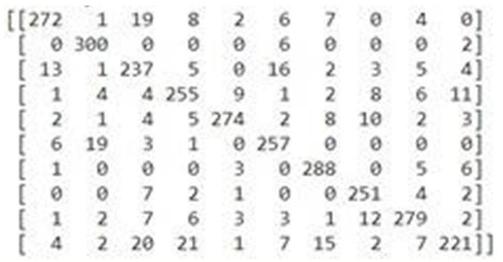
Area under Curve Percentage of 79.20946760761198

4.4: RandomForestClassifier

RandomForestClassifier(n_estimators=4000, max_depth=40, random_state=0)

n_estimators(=4000) tells us the maximum number of trees to be formed for extracting best results and mx_depth(=40) gives us the maximum depth of the tree.

Confusion Matrix:



```
[[272   1  19   8   2   6   7   0   4   0]
 [  0 300   0   0   0   6   0   0   0   2]
 [ 13   1 237   5   0  16   2   3   5   4]
 [  1   4   4 255   9   1   2   8   6  11]
 [  2   1   4   5 274   2   8  10   2   3]
 [  6  19   3   1   0 257   0   0   0   0]
 [  1   0   0   0   3   0 288   0   5   6]
 [  0   0   7   2   1   0   0 251   4   2]
 [  1   2   7   6   3   3   1  12 279   2]
 [  4   2  20  21   1   7  15   2   7 221]]
```

This method gave an

Accuracy of 87.88788788788789

Precision of 87.8625894183172

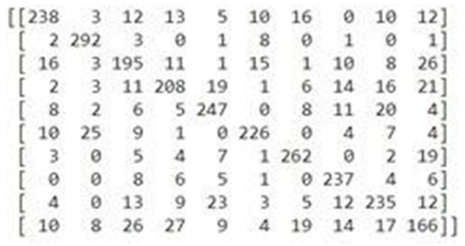
F1 Score of 87.77180515921353

Area under Curve Percentage of 98.88706249565787

4.5: MLPClassifier

MLPClassifier(random_state=1, max_iter=300)

Confusion Matrix:



```
[[238   3  12  13   5  10  16   0  10  12]
 [  2 292   3   0   1   8   0   1   0   1]
 [ 16   3 195  11   1  15   1  10   8  26]
 [  2   3  11 208  19   1   6  14  16  21]
 [  8   2   6   5 247   0   8  11  20   4]
 [ 10  25   9   1   0 226   0   4   7   4]
 [  3   0   5   4   7   1 262   0   2  19]
 [  0   0   8   6   5   1   0 237   4   6]
 [  4   0  13   9  23   3   5  12 235  12]
 [ 10   8  26  27   9   4  19  14  17 166]]
```

This method gave an

Accuracy of 76.9436102769436

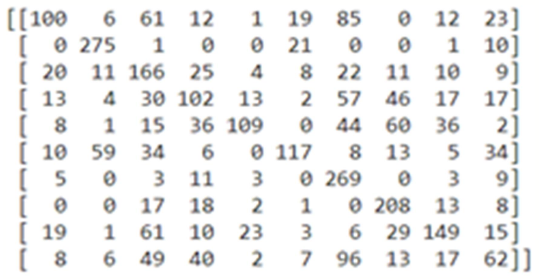
Precision of 76.68040399591673

F1 Score of 76.74536109036163

Area under Curve Percentage of 96.88831333743879

4.6: GaussianNB()

Confusion Matrix:



```
[[100   6  61  12   1  19  85   0  12  23]
 [  0 275   1   0   0  21   0   0   1  10]
 [ 20  11 166  25   4   8  22  11  10   9]
 [ 13   4  30 102  13   2  57  46  17  17]
 [  8   1  15  36 109   0  44  60  36   2]
 [ 10  59  34   6   0 117   8  13   5  34]
 [  5   0   3  11   3   0 269   0   3   9]
 [  0   0  17  18   2   1   0 208  13   8]
 [ 19   1  61  10  23   3   6  29 149  15]
 [  8   6  49  40   2   7  96  13  17  62]]
```

This method gave an

Accuracy of 51.95195195195195

Precision of 53.2797814824036

F1 Score of 50.267561071491414

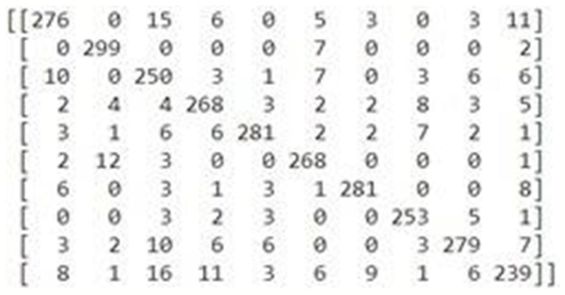
Area under Curve Percentage of 88.45643124480602

This model gave the least accuracy and predicted only 51.9% of the values correctly. Though the area under curve percentage is fairly decent, the other parameters are not up to the mark.

4.7: XGBClassifier

XGBClassifier(n_estimators=3500, learning_rate=0.05,base_score=1)

Confusion Matrix:



```
[[276   0  15   6   0   5   3   0   3  11]
 [  0 299   0   0   0   7   0   0   0   2]
 [ 10   0 250   3   1   7   0   3   6   6]
 [  2   4   4 268   3   2   2   8   3   5]
 [  3   1   6   6 281   2   2   7   2   1]
 [  2  12   3   0   0 268   0   0   0   1]
 [  6   0   3   1   3   1 281   0   0   8]
 [  0   0   3   2   3   0   0 253   5   1]
 [  3   2  10   6   6   0   0   3 279   7]
 [  8   1  16  11   3   6   9   1   6 239]]
```

This method gave an

Accuracy of 89.88988988988989

Precision of 60.30170588040333

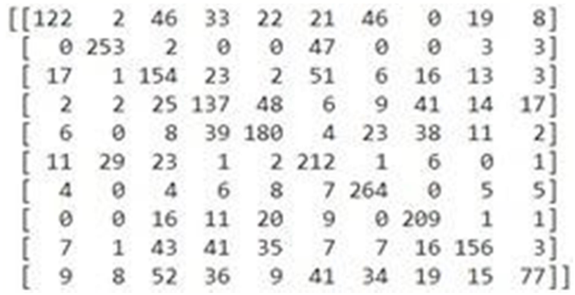
F1 Score of 89.88252136756282

Area under Curve Percentage of 99.27071784913669

4.8: XGBRFClassifier

XGBRFClassifier(objective= 'multi:softmax')

Confusion Matrix:



```
[[122   2  46  33  22  21  46   0  19   8]
 [  0 253   2   0   0  47   0   0   3   3]
 [ 17   1 154  23   2  51   6  16  13   3]
 [  2   2  25 137  48   6   9  41  14  17]
 [  6   0   8  39 180   4  23  38  11   2]
 [ 11  29  23   1   2 212   1   6   0   1]
 [  4   0   4   6   8   7 264   0   5   5]
 [  0   0  16  11  20   9   0 209   1   1]
 [  7   1  43  41  35   7   7  16 156   3]
 [  9   8  52  36   9  41  34  19  15  77]]
```

This method gave an

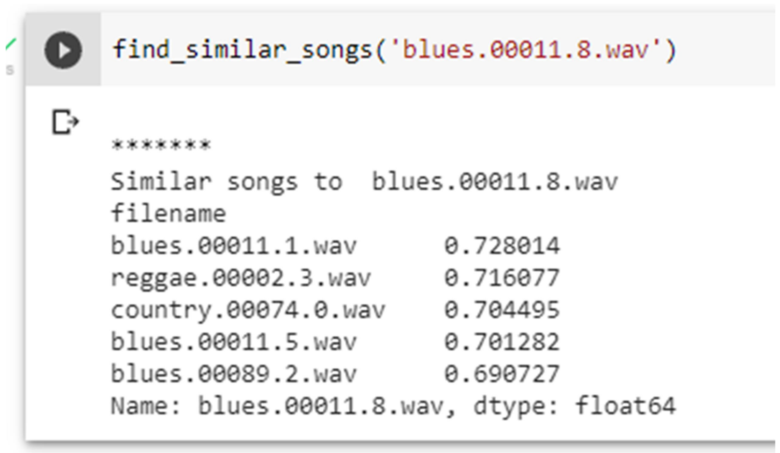
Accuracy of 58.85885885885885

Precision of 58.85885885885885

F1 Score of 57.86900433914437

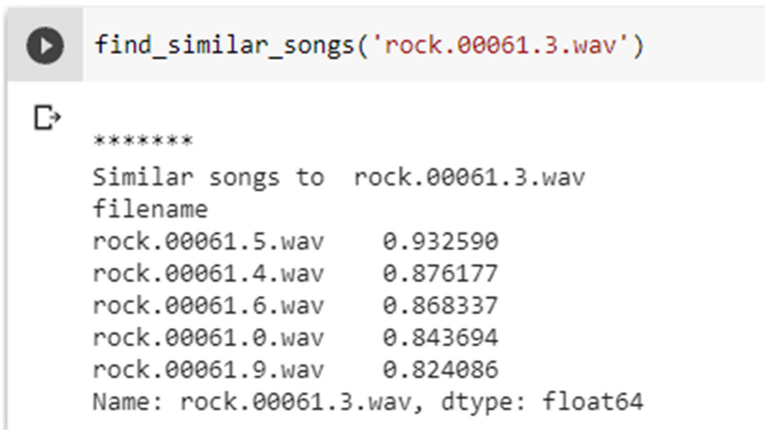
Area under Curve Percentage of 91.1185219832134

4.9: Music Recommendation :



```
find_similar_songs('blues.00011.8.wav')

*******
Similar songs to  blues.00011.8.wav
filename
blues.00011.1.wav      0.728014
reggae.00002.3.wav     0.716077
country.00074.0.wav    0.704495
blues.00011.5.wav      0.701282
blues.00089.2.wav      0.690727
Name: blues.00011.8.wav, dtype: float64
```

Five similar songs to audio ‘blues.00011.8’ are displayed.

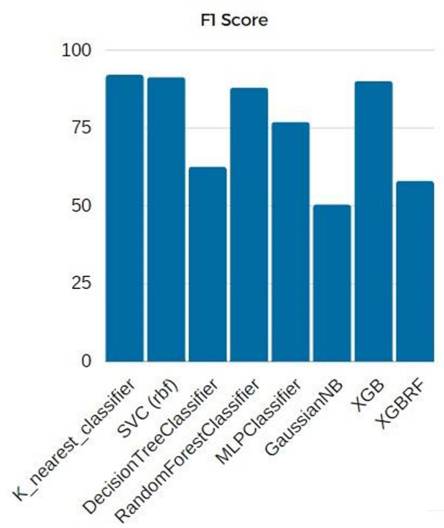


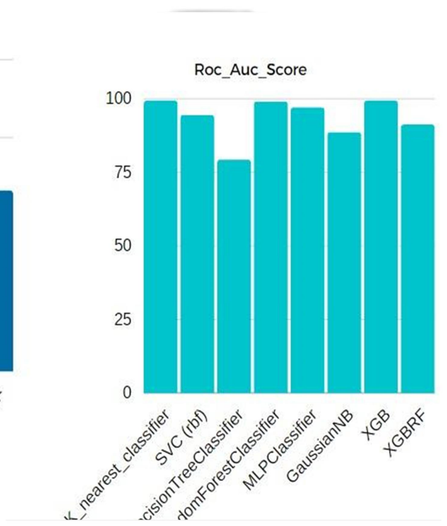
```
find_similar_songs('rock.00061.3.wav')

*******
Similar songs to  rock.00061.3.wav
filename
rock.00061.5.wav    0.932590
rock.00061.4.wav    0.876177
rock.00061.6.wav    0.868337
rock.00061.0.wav    0.843694
rock.00061.9.wav    0.824086
Name: rock.00061.3.wav, dtype: float64
```

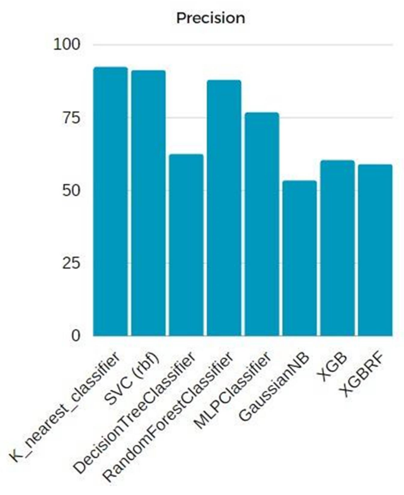
Five similar songs to audio ‘rock.00016.3’ are displayed.

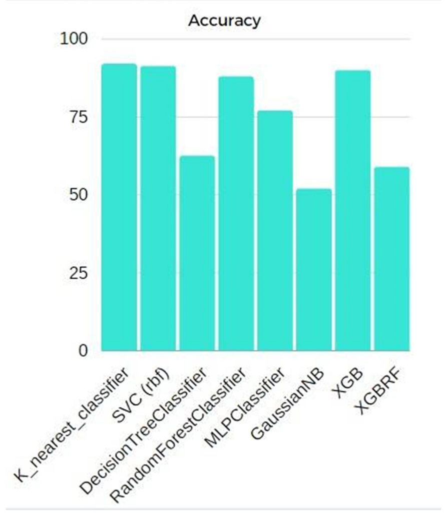
4.10: Visual Comparison using Graphs

The following graphs compare the various factors that were discussed above as well:









Therefore, the best accuracy is given by K_Neighbors and XGB Classifiers.

The above graphs and results show that.

CONCLUSION AND FUTURE WORKS

This research work provides the details of music classification and recommendations using machine learning techniques. Each track from GTZAN dataset is obtained. This is done by librosa package of python. A piece of software is implemented which performs classification of large datasets of songs into their respective genres.

The extension of this work would be considered bigger data sets and also track in different data formats. Also with time the style represented by each genre will continue to change. So the objective for future will be to stay updated with the change in styles of genre and classification and extending our software to work on these updated styles. This work can also be extended to work as a music recommendations system depending on mood of person.

BIBLIOGRAPHY

1. 'https://librosa.org/doc/latest/index.html' for reading music files and getting its features
2. 'https://scikit-learn.org/stable/supervised_learning.html#supervised-learning' for using all the classification algorithms and getting all the results.
3. 'https://xgboost.readthedocs.io/en/latest/' used for a specific classification model.
4. 'https://pandas.pydata.org/pandas-docs/stable/index.html' used to read datas, differentiate them to independent and dependent features and scaling of datas.
5. 'https://numpy.org/doc/'
6. 'https://matplotlib.org/stable/contents.html' used for plotting output graphs with results.26