GENRE CLASSIFICATION FROM SONG FEATURES

QMBU 450 - Final Project

KORAY YENAL

0049622

Introduction

Over the recent years, music streaming services have gained huge popularity. They've become part of our daily lives. Services such as Spotify or Youtube Music have started to offer millions of songs in their libraries. For that reason, it has become a crucial task for music streaming services to categorize each song into their corresponding genre. This allows for streaming services to make genre-specific playlists or give personalized recommendations to users.

This term project analyzes datasets from a music intelligence group, The Echo Nest (Now owned by Spotify). The project's goal is to classify songs into two genres - Hip-Hop and Rock, with the help of musical features, such as tempo or acousticness.

The project uses two sets of data: First dataset is about the song's specific features such as title, artist, number of listens or duration. It is in the form of CSV. The other dataset is about the track metrics (musical features) that implies the song's genre: energy, tempo, liveness, valence, acousticness etc. The second dataset is in the form of JSON. We merge these two datasets' relevant columns and analyze the merged dataset itself.

The project achieves following things in the following order:

- Data Cleaning
- Exploratory Data Analysis - Correlation Matrix, Standardization
- Dimensionality Reduction - PCA, cumulative explained ratio
- Performance Measurement - Decision Trees, Logistic Regression,
- Data Balancing to improve the performance of Hip-Hop classification
- Further Performance Measurement - SVM, Neural Nets
- Model Evaluation - K-fold Cross Validation

1.Data Importing and Initial Look

We start by loading both sets of datas. We create two panda Dataframes out of them, so that we can merge these datasets.

songs = pd.read_csv('/Users/korayyenal/Downloads/echonestdataset.csv')

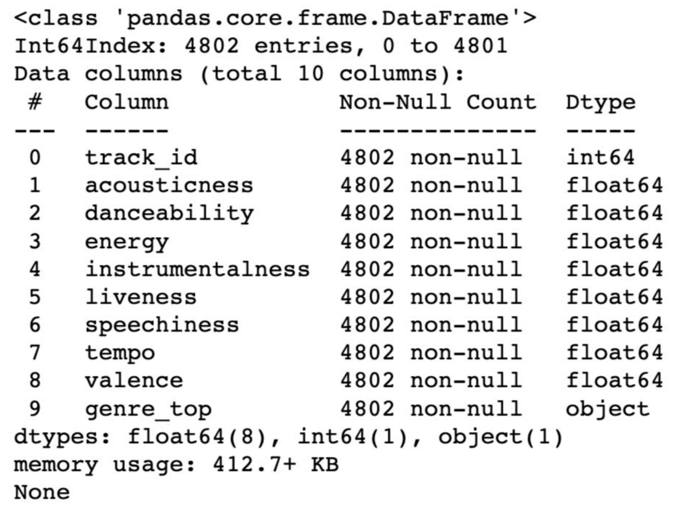
metrics = pd.read_json('/Users/korayyenal/Downloads/metrics.json', precise_float=True)

Then we merge the relevant columns, taking all the metrics from the JSON and taking 'track_id' and 'genre_top' columns from the CSV file.

merged_songs = pd.merge(metrics, songs[['track_id', 'genre_top']], on='track_id')

We examine our merged dataset by .info(). The resulting table is below:

print(merged_songs.info())



```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4802 entries, 0 to 4801
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_id          4802 non-null   int64
 1   acousticness      4802 non-null   float64
 2   danceability      4802 non-null   float64
 3   energy            4802 non-null   float64
 4   instrumentalness  4802 non-null   float64
 5   liveness          4802 non-null   float64
 6   speechiness       4802 non-null   float64
 7   tempo             4802 non-null   float64
 8   valence           4802 non-null   float64
 9   genre_top         4802 non-null   object
dtypes: float64(8), int64(1), object(1)
memory usage: 412.7+ KB
None
```

We see that there is 8 musical features, or metrics, namely:

1. Acousticness
2. Danceability
3. Energy
4. Instrumentalness

5. Liveness
6. Speechiness
7. Tempo
8. Valence

They are all type ‘float’, therefore represented as a real number for each song.
In the next step, we will examine these features in more detail.

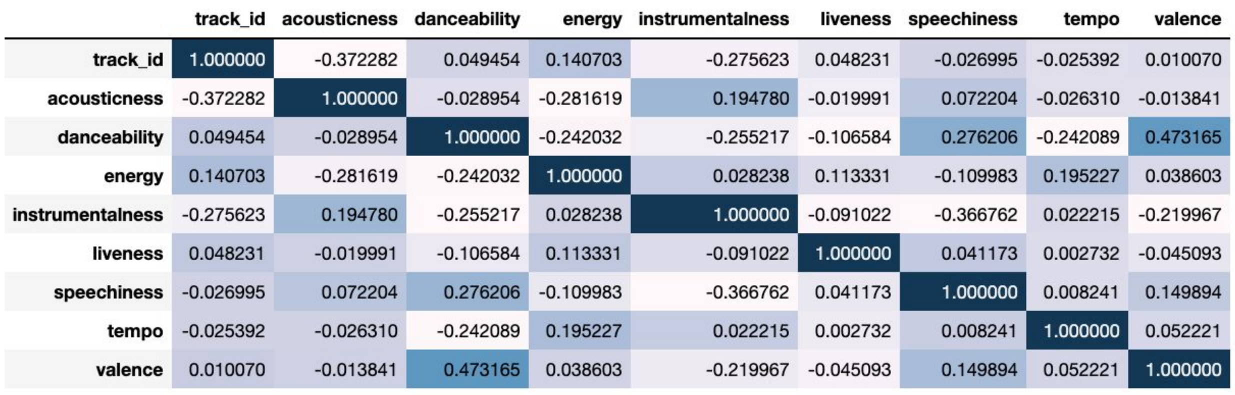
2. Pairwise relationships between variables via Correlation Matrix

While it is important that we have features that can explain a song’s genre, it is also crucial that we check for any strong correlations between each feature. Our aim is to keep the model simple but not underfitting. Similarly, we need to use just enough variables for interpretability but avoid overfitting. Therefore, we need to avoid the redundancy of explanatory variables. Therefore, we use the **correlation matrix** to check whether there is any strong correlation among features.

We use panda’s built-in function to get the correlation matrix.

corr_metrics = merged_songs.corr()

corr_metrics.style.background_gradient()



	track_id	acousticness	danceability	energy	instrumentalness	liveness	speechiness	tempo	valence
track_id	1.000000	-0.372282	0.049454	0.140703	-0.275623	0.048231	-0.026995	-0.025392	0.010070
acousticness	-0.372282	1.000000	-0.028954	-0.281619	0.194780	-0.019991	0.072204	-0.026310	-0.013841
danceability	0.049454	-0.028954	1.000000	-0.242032	-0.255217	-0.106584	0.276206	-0.242089	0.473165
energy	0.140703	-0.281619	-0.242032	1.000000	0.028238	0.113331	-0.109983	0.195227	0.038603
instrumentalness	-0.275623	0.194780	-0.255217	0.028238	1.000000	-0.091022	-0.366762	0.022215	-0.219967
liveness	0.048231	-0.019991	-0.106584	0.113331	-0.091022	1.000000	0.041173	0.002732	-0.045093
speechiness	-0.026995	0.072204	0.276206	-0.109983	-0.366762	0.041173	1.000000	0.008241	0.149894
tempo	-0.025392	-0.026310	-0.242089	0.195227	0.022215	0.002732	0.008241	1.000000	0.052221
valence	0.010070	-0.013841	0.473165	0.038603	-0.219967	-0.045093	0.149894	0.052221	1.000000

At first glance, we see that valence and danceability are somewhat positively correlated compared to the other pairs, with 0.47 value. So we could say that as a song’s valence is high, i.e. the song is more positive, the song tends to get more danceable. Similarly, instrumentalness and speechiness are negatively correlated with -0.36 value, which is not so small. However, these values do not justify any strong correlation between pairs.

Although we checked the correlation matrix, we could not find significant correlation between any pair.

3. Standardizing the data

Next step is to apply the common approach **Principal Component Analysis (PCA)** to understand the relative contribution of each metric towards the variance. We will check if data can be explained with a smaller number of metrics.

To do that, first we need to apply **standardization**, since some metrics are not in the same value range. For instance, ‘Tempo’ metric values are somewhere around 100, which is typical for BPM unit (Beats per Minute). We need to standardize such metrics, so that all resulting metrics have a **mean** of 0 and a **standard deviation** of 1. If we do not do this, metrics with larger values will have more weight and therefore create bias in the approach.

```
song_feats = merged_songs.drop(['track_id', 'genre_top'], axis=1)

labels = merged_songs['genre_top']

scaler = StandardScaler()

scaled_train_feats = scaler.fit_transform(song_feats)
```

4. PCA on Scaled data using Scree Plots and Cumulative Explained Ratio plot

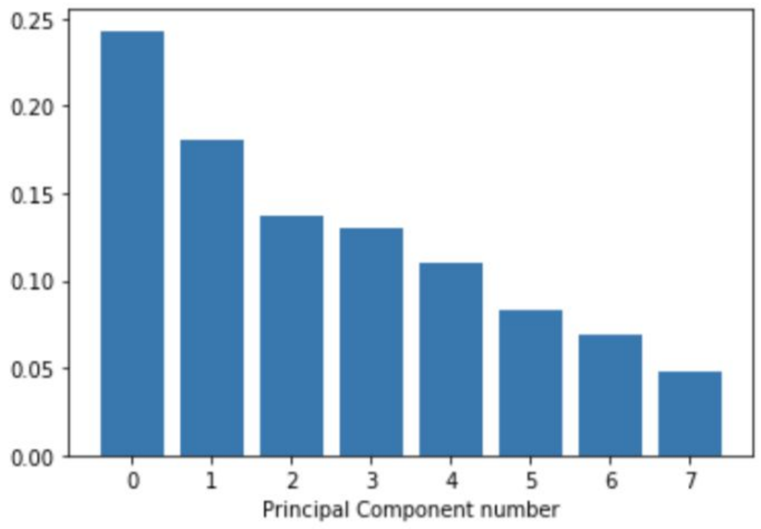
Now that our data is standardized, we can apply PCA to see to what extent we can reduce the dimensionality. Applying PCA to our dataset will speed up the computation time, since our dataset is quite large. In this part, we use two things: **scree-plot** and **cumulative explained ratio** plot.

```
%matplotlib inline

pca = PCA()

pca.fit(scaled_train_feats)

exp_variance = pca.explained_variance_ratio_

fig, ax = plt.subplots()

ax.bar(range(pca.n_components_), exp_variance)

ax.set_xlabel('Principal Component number')
```

For the scree-plot, we look for a sharp drop between any data point to decide the cut-off point.

Scree-plot Result:

```
Text(0.5, 0, 'Principal Component number')
```



However, there does not seem to be a sharp drop in any number, except the first two. It is obvious that we cannot reduce the number to 0 or 1, so we neglect that drop.

We move on to check the **cumulative explained ratio.** Arbitrarily, we set the level to 0.90 variance.

cum_exp_variance = np.cumsum(exp_variance)

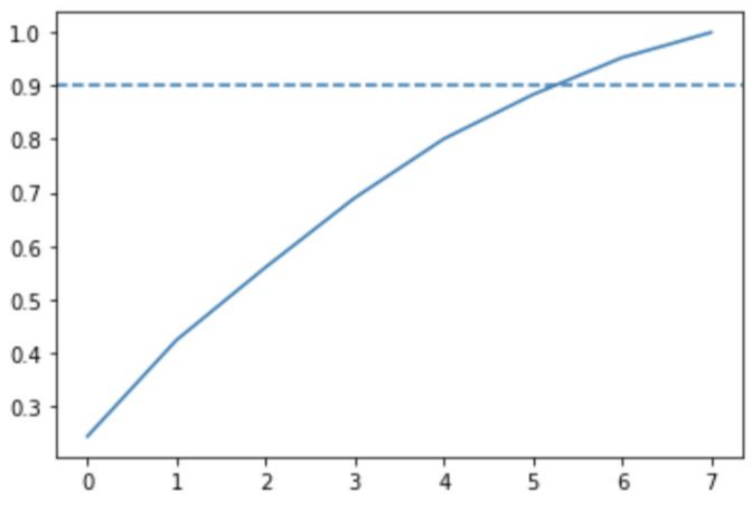
fig, ax = plt.subplots()

ax.plot(cum_exp_variance)

ax.axhline(y=0.9, linestyle='--')

n_components = 6

Cumulative Explained Varience Result:



The intersection says that we can pick **6 components** to explain 90% of variance. We don't pick 5 here because it falls below the 90% threshold that we set previously.

We continue with applying PCA with this many components.

pca = PCA(n_components, random_state=10)

pca.fit(scaled_train_feats)

pca_projection = pca.transform(scaled_train_feats)

5. Decision Tree for genre classification

With lower dimensional data with PCA applied, we can classify each song into their corresponding genre.

To start, we split the data into **train** and **test**.

train_feats, test_feats, train_labels, test_labels = train_test_split(pca_projection, labels, random_state=10)

Then we apply **Decision Tree** algorithm to see its performance.

```
tr = DecisionTreeClassifier(random_state=10)
tr.fit(train_feats, train_labels)

predict_tr = tree.predict(test_feats)
tr.score(test_feats, test_labels)
```

Decision tree score: **0.8434**

Decision Tree's performance is not bad!

6. Decision tree comparison to logistic regression

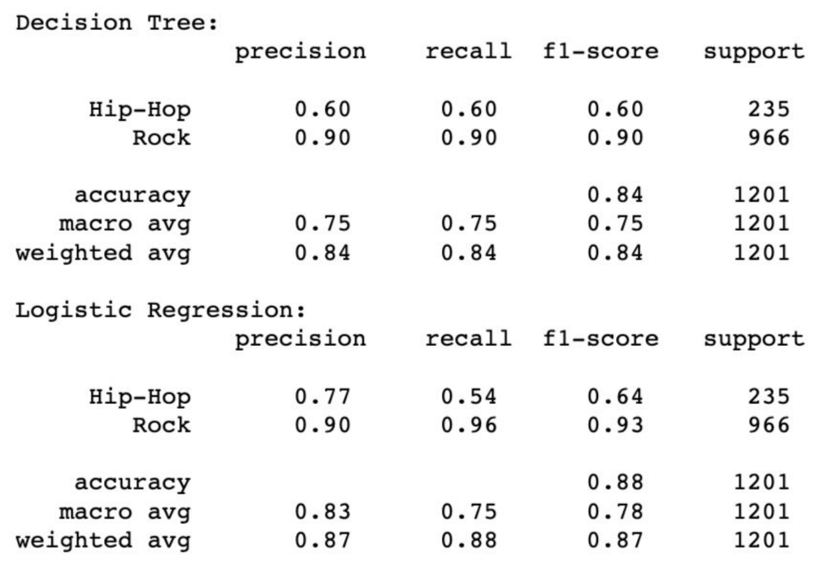
Although we get a decent result, it is better to compare it with other methods, such as logistic regression. We use logistic regression here because it's a simple but effective algorithm.

```
lr = LogisticRegression(random_state=10)
lr.fit(train_feats, train_labels)
predict_lr = lr.predict(test_feats)
```

We compare the results via a classification report, to see false positives and false negatives of each.

```
classification_dt = classification_report(test_labels, predict_dt)
classification_lr = classification_report(test_labels, predict_lr)


print("Decision Tree: \n", classification_dt)
print("Logistic Regression: \n", classification_lr)
```

Resulting Table:



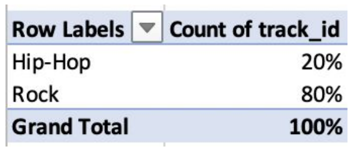
```
Decision Tree:
              precision    recall  f1-score   support

     Hip-Hop       0.60      0.60      0.60       235
        Rock       0.90      0.90      0.90       966

    accuracy                           0.84      1201
   macro avg       0.75      0.75      0.75      1201
weighted avg       0.84      0.84      0.84      1201

Logistic Regression:
              precision    recall  f1-score   support

     Hip-Hop       0.77      0.54      0.64       235
        Rock       0.90      0.96      0.93       966

    accuracy                           0.88      1201
   macro avg       0.83      0.75      0.78      1201
weighted avg       0.87      0.88      0.87      1201
```

Logistic Regression gave 3% better ‘precision’ overall.

However, looking at ‘recall’, Hip-Hop songs were wrongly classified as Rock disproportionately. This might be because there is an imbalance in terms of number of data points. Rock has far more data points than hip hop, therefore we need to balance that as the next step.

7. Data Balancing

When we take the pivot of Data , we see the following result:



Row Labels	Count of track_id
Hip-Hop	20%
Rock	80%
Grand Total	**100%**

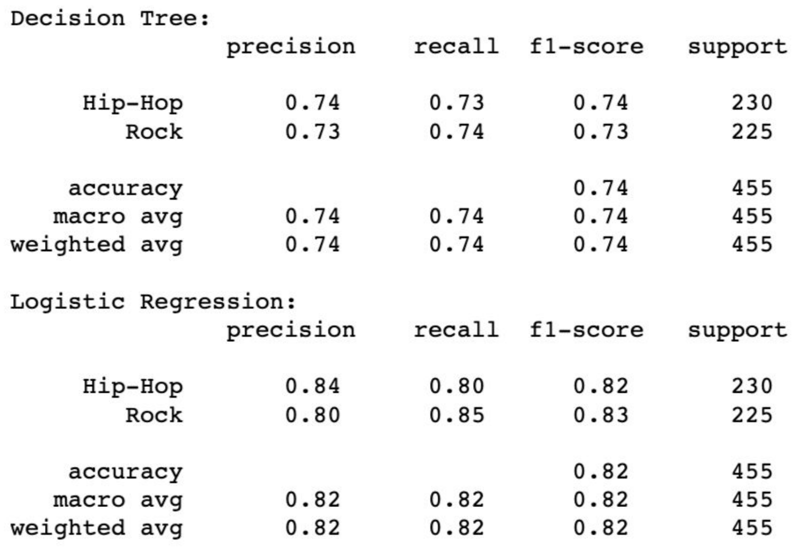
We have much more data points for rock songs than for hip-hop (80%-20% respectively), therefore the model can distinguish rock songs much more accurately than hip hop because of data abundance.

To overcome this, we sample equal number of songs for both Rock and Hip-Hop genres.

```
hiphopsongs = merged_songs[merged_songs['genre_top']=='Hip-Hop']

rocksongs = merged_songs[merged_songs['genre_top']=='Rock']

#Sampling equal numbers from both

rocksongs = rocksongs.sample(len(hiphopsongs), random_state=10)

hiphop_rock_merged = pd.concat([rocksongs, hiphopsongs])

song_feats = hiphop_rock_merged.drop(['genre_top', 'track_id'], axis=1)

labels = hiphop_rock_merged['genre_top']

pca_projection = pca.fit_transform(scaler.fit_transform(song_feats))

# Redefine train and test with the balanced data

train_feats, test_feats, train_labels, test_labels = train_test_split(pca_projection, labels,
random_state=10)

# Train Decision Tree on the balanced

dt = DecisionTreeClassifier(random_state=10)
```

```
dt.fit(train_feats, train_labels)

predict_dt = dt.predict(test_feats)

# Train LR on the balanced

lr = LogisticRegression(random_state=10)

lr.fit(train_feats, train_labels)

predict_lr = lr.predict(test_feats)

# Compare two models

print("Decision Tree: \n", classification_report(test_labels, predict_dt))

print("Logistic Regression: \n", classification_report(test_labels, predict_lr))
```

Results Table after Balancing:



```
Decision Tree:
              precision    recall  f1-score   support

     Hip-Hop       0.74      0.73      0.74       230
        Rock       0.73      0.74      0.73       225

    accuracy                           0.74       455
   macro avg       0.74      0.74      0.74       455
weighted avg       0.74      0.74      0.74       455

Logistic Regression:
              precision    recall  f1-score   support

     Hip-Hop       0.84      0.80      0.82       230
        Rock       0.80      0.85      0.83       225

    accuracy                           0.82       455
   macro avg       0.82      0.82      0.82       455
weighted avg       0.82      0.82      0.82       455
```

As a result, we removed the bias of the more prevalent class, the rock genre. As you can see, the precision and recall performances are similar, which is a success!

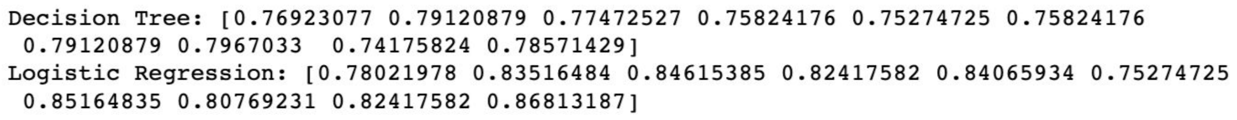
8.Cross-Validation for Model Evaluation

For model evaluation, we use k-fold cross validation. We shuffle the data for randomness, and split the data into 10 folds.

Finally, we aggregate the results for each fold to show the total model performance.

```
kf = KFold(10,shuffle=True, random_state=10)
dt = DecisionTreeClassifier(random_state=10)
lr = LogisticRegression(random_state=10)

dt_score = cross_val_score(dt, pca_projection, labels, cv=kf)
lr_score = cross_val_score(lr, pca_projection, labels, cv=kf)

print("Decision Tree:", tree_score, "\nLogistic Regression:", logit_score)
```

Results:



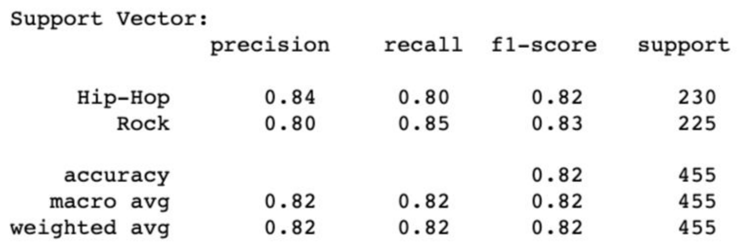
```
Decision Tree: [0.76923077 0.79120879 0.77472527 0.75824176 0.75274725 0.75824176
 0.79120879 0.7967033  0.74175824 0.78571429]
Logistic Regression: [0.78021978 0.83516484 0.84615385 0.82417582 0.84065934 0.75274725
 0.85164835 0.80769231 0.82417582 0.86813187]
```

9. Support Vector Machines

As a further comparison, we added SVM as a learning model to see its performance on the balanced data.

```
clf = svm.SVC(kernel='linear') # Linear Kernel
train_feats, test_feats, train_labels, test_labels = train_test_split(pca_projection, labels,
random_state=10)
#Train the model using the training sets
clf.fit(train_feats, train_labels)
predict_svm = clf.predict(test_feats)
classification_svm = classification_report(test_labels, predict_svm)
print("Support Vector: \n", classification_report(test_labels, predict_svm))
```

The resulting table:

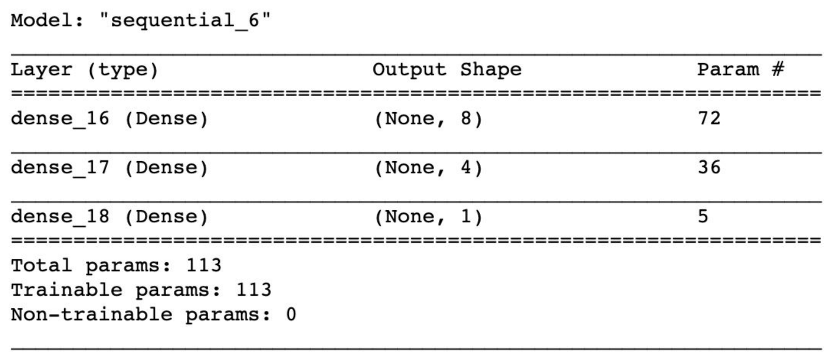


```
Support Vector:
               precision    recall  f1-score   support

     Hip-Hop       0.84      0.80      0.82       230
        Rock       0.80      0.85      0.83       225

    accuracy                           0.82       455
   macro avg       0.82      0.82      0.82       455
weighted avg       0.82      0.82      0.82       455
```

10.Neural Nets

As the last model, we try MLP Classifier. We use binary_crossentropy for determining whether the song is Rock or Hip-Hop. As an optimizer, we use “ADAM” optimization algorithm.

8 input layer,
4 hidden layer
1 output layer is used.



```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 8)                 72
_________________________________________________________________
dense_17 (Dense)             (None, 4)                 36
_________________________________________________________________
dense_18 (Dense)             (None, 1)                 5
=================================================================
Total params: 113
Trainable params: 113
Non-trainable params: 0
_________________________________________________________________
```

model.compile(loss='binary_crossentropy',
optimizer='adam',
metrics=['accuracy'])

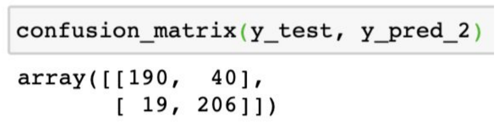
model.fit(X_train, y_train,epochs=8, batch_size=1, verbose=1)

y_pred = model.predict(X_test)

score = model.evaluate(X_test, y_test,verbose=1)

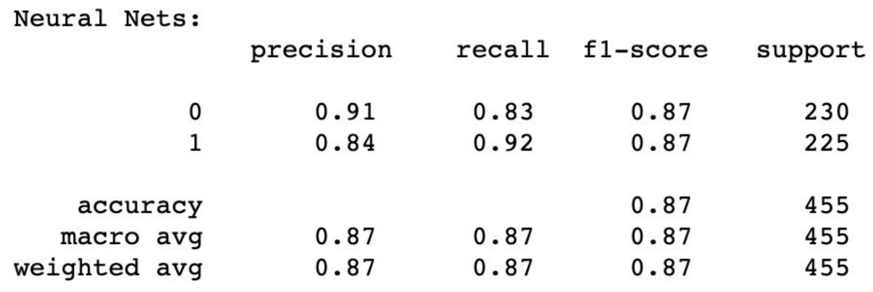
The Resulting Confusion Matrix is as follows:

confusion_matrix(y_test, y_pred_2)



```
confusion_matrix(y_test, y_pred_2)

array([[190,  40],
       [ 19, 206]])
```

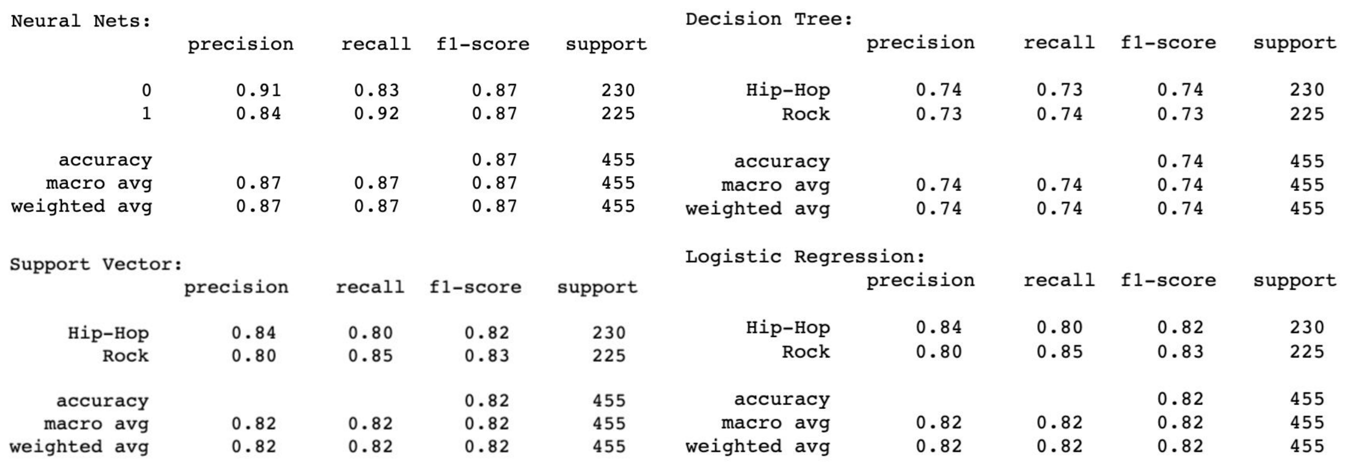
The resulting Classification Report for Neural Network is as follows:



```
Neural Nets:
              precision    recall  f1-score   support

           0       0.91      0.83      0.87       230
           1       0.84      0.92      0.87       225

    accuracy                           0.87       455
   macro avg       0.87      0.87      0.87       455
weighted avg       0.87      0.87      0.87       455
```

We define 1 as Rock and 0 as Hip-Hop. Since we balanced the data, the weighted average for both got the same results, which is what we wanted.

Resulting Tables Together:



```
Neural Nets:
              precision    recall  f1-score   support

           0       0.91      0.83      0.87       230
           1       0.84      0.92      0.87       225

    accuracy                           0.87       455
   macro avg       0.87      0.87      0.87       455
weighted avg       0.87      0.87      0.87       455
```

```
Decision Tree:
              precision    recall  f1-score   support

     Hip-Hop       0.74      0.73      0.74       230
        Rock       0.73      0.74      0.73       225

    accuracy                           0.74       455
   macro avg       0.74      0.74      0.74       455
weighted avg       0.74      0.74      0.74       455
```

```
Support Vector:
              precision    recall  f1-score   support

     Hip-Hop       0.84      0.80      0.82       230
        Rock       0.80      0.85      0.83       225

    accuracy                           0.82       455
   macro avg       0.82      0.82      0.82       455
weighted avg       0.82      0.82      0.82       455
```

```
Logistic Regression:
              precision    recall  f1-score   support

     Hip-Hop       0.84      0.80      0.82       230
        Rock       0.80      0.85      0.83       225

    accuracy                           0.82       455
   macro avg       0.82      0.82      0.82       455
weighted avg       0.82      0.82      0.82       455
```

11.Conclusion

As a result, we successfully applied 4 different models to our dataset and got the results. As we can see from the resulting tables, the MLP classifier got the most accurate results. SVM and Logistic Regression had similar accuracy results. Decision Tree got the weakest prediction.

All in all, with a detailed analysis, we can predict the songs genre with 87% overall accuracy if Neural Nets selected.

References:

https://www.datacamp.com/community/tutorials/deep-learning-python#modeling