Al-Khwarizmi

Institute of Science, Technology And Innovation

Project report

Music Genre Classification Using Machine Learning

By:
Brahim Erraji

Under the supervision of:
Pr. Chairi Ikram And
Pr. Zerouaoui Hasnae

University Year 2020/2021

Acknowledgement

I would like to express my special thanks and gratitude to the professeurs **Ikram Chairi** and **Hasnae Zerouaoui**, for their guidance and support throughout the artificial intelligence course.
Also I would like to thank the folks behind the GTZAN dataset for releasing it freely for the purpose of research and developement.

Contents

1 Introduction

In the context of the artificial intelligence course that I took in my third year of bachelor's degree, We had to apply the skills and tools learned in a real world project. I chose to work on the *music genre classification problem*. The challenge is to assign correctly one genre from nine to a given sound clip.

2 Project Presentation

2.1 The need for a music classifier

Music nowadays is ubiquitous, and consequently a good music classifier is needed now more than ever. The applications are truly endless:

- Music recommendation (Spotify, Youtube Music)
- Music generation
- ...

2.2 Project goals

The final program should be able to:

- Extract features from a given sound clip
- Correctly assign one genre from nine possible genres

3 Dataset

3.1 Dataset description

The dataset used is this project GTZAN dataset. GTZAN is the most-used public dataset for evaluation in machine listening research for music genre recognition (MGR). The files were collected in 2000-2001 from a variety of sources including personal CDs, radio, microphone recordings.
It contains over 9990 sound clips with the same duration 30 seconds, each clips has a label (genre), which makes it a supervised learning problem.

The sound excerpts are digital audio files in .wav format. Sound waves are digitized by sampling them at discrete intervals known as the sampling rate (typically 44.1kHz for CD-quality audio meaning samples are taken 44,100 times per second).

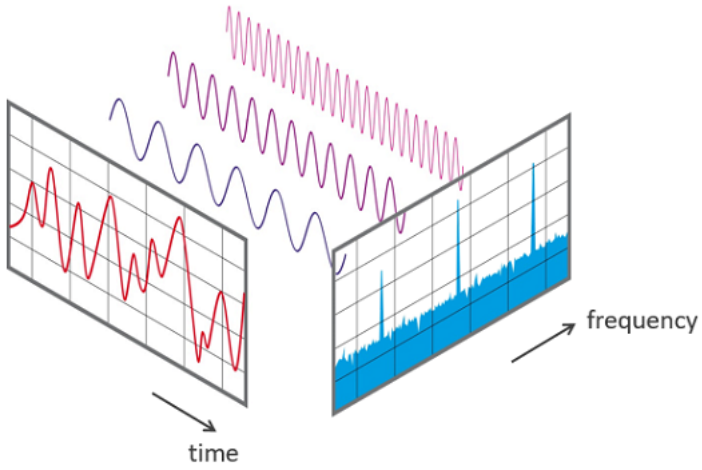
3.2 Features

Each sound clip has a set of features (Chroma feature, Audio power, Spectral centroid, Tempo ..).
The GTZAN dataset has 58 sound features, all numerical.
We can mention the following features:

- chroma stft mean
- rms mean
- spectral centroid mean
- spectral bandwidth mean
- ...

3.2.1 Feature Visualization

Sound is represented in the form of an audio signal having parameters such as frequency, bandwidth, decibel, etc. A typical audio signal can be expressed as a function of Amplitude and Time shown below:



3.2.2 Feature extraction

In Python, we can extract feature using the library Librosa.

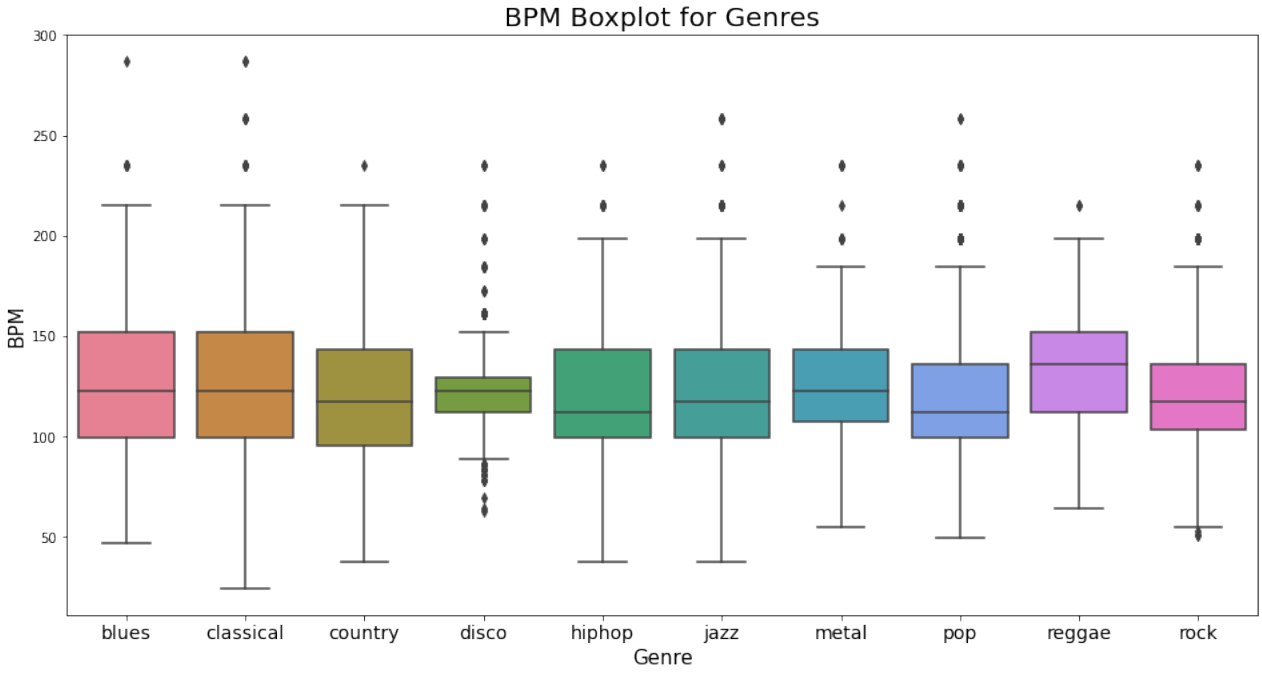
Feature extraction from Audio signal

```
In [235]: audio_path = 'genres/blues/blues.00001.wav'
          x,sr = librosa.load(audio_path)

In [124]: spectral_centroids = librosa.feature.spectral_centroid(x, sr=sr)[0]

In [125]: spectral_centroids.mean()
Out[125]: 1530.1766787460795

In [126]: spectral_centroids.var()
Out[126]: 375850.0736486866

In [127]: chroma_stft = librosa.feature.chroma_stft(x,sr=sr)[0]

In [128]: rms = librosa.feature.rms(x)

In [129]: rms[0]
Out[129]: array([0.06771811, 0.10716628, 0.14316952, ..., 0.13760294, 0.1339183 ,
                 0.12976366], dtype=float32)

In [130]: spectral_bandwith = librosa.feature.spectral_bandwidth(x)

In [131]: len(spectral_centroids)
Out[131]: 1293
```

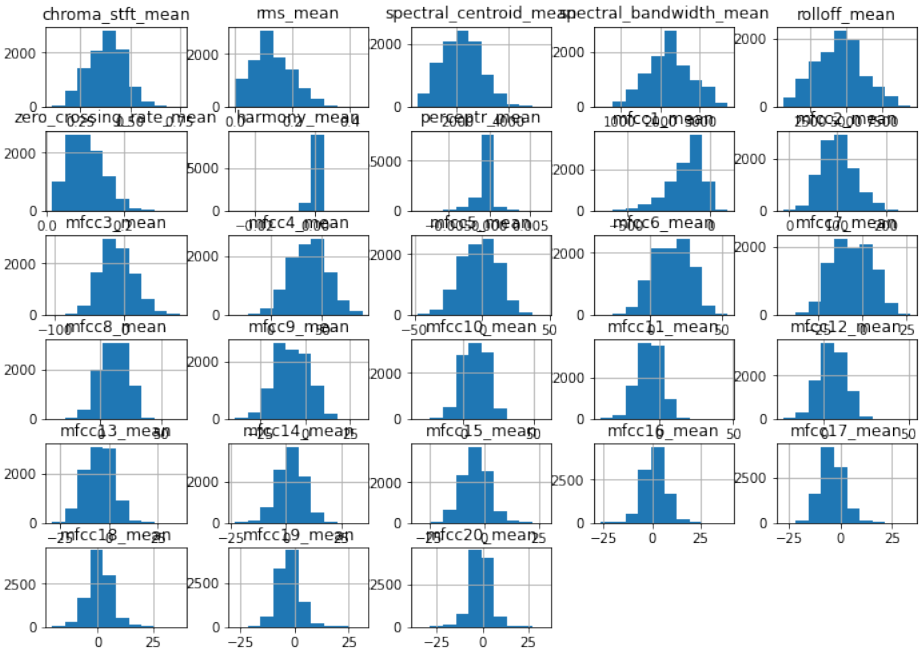
4 Classification

4.1 Data visualization

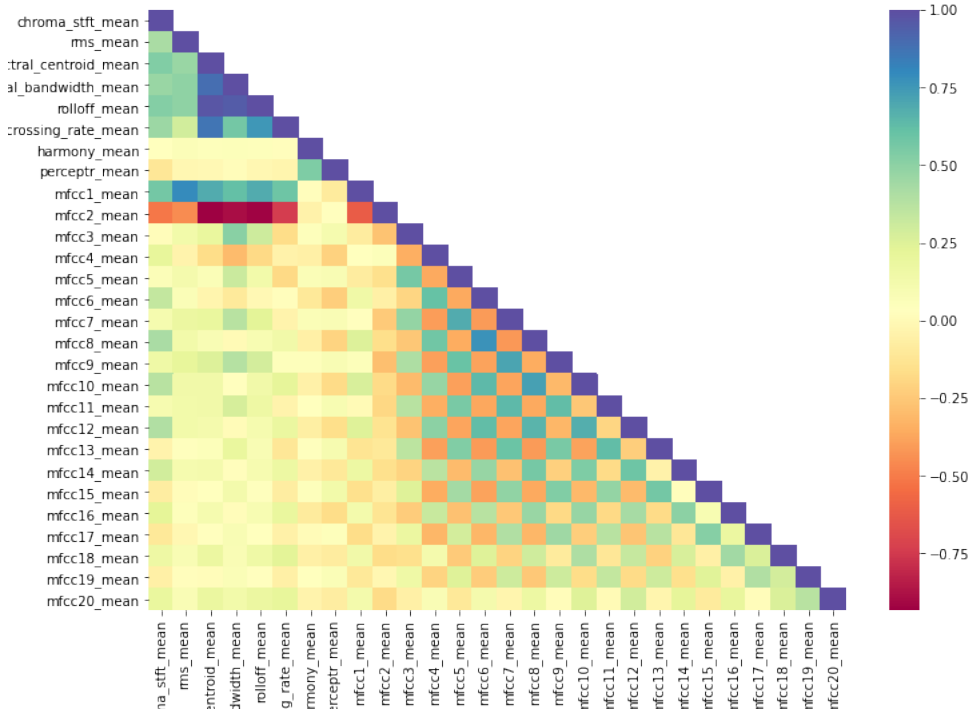
To understand our dataset, it is useful to sees visually the distribution of some of the features:



It is also important to see the distribution of each variable seperately:



Another useful plot is the correlation matrix, it allows us to clearly see if there are dependances between the features, which might cause problems in the training phase:


chroma_stft_mean
rms_mean
:tral_centroid_mean
al_bandwidth_mean
rolloff_mean
:rossing_rate_mean
harmony_mean
perceptr_mean
mfcc1_mean
mfcc2_mean
mfcc3_mean
mfcc4_mean
mfcc5_mean
mfcc6_mean
mfcc7_mean
mfcc8_mean
mfcc9_mean
mfcc10_mean
mfcc11_mean
mfcc12_mean
mfcc13_mean
mfcc14_mean
mfcc15_mean
mfcc16_mean
mfcc17_mean
mfcc18_mean
mfcc19_mean
mfcc20_mean
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75

4.2 Data exploring

Dataset has two columns that are irrelevant to the genre of the music : *filename* and *length*.

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mea
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.74438
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.05150
2	blues.00000.2.wav	66149	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.56513
3	blues.00000.3.wav	66149	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.03998
4	blues.00000.4.wav	66149	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.50388

Therefore they should be dropped:

	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mean	spectral_bandwidth_var
0	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.744388	117335.771563
1	0.343065	0.086147	0.112699	0.001450	1816.693777	90525.690866	2010.051501	65671.875673
2	0.346815	0.092243	0.132003	0.004620	1788.539719	111407.437613	2084.565132	75124.921716
3	0.363639	0.086856	0.132565	0.002448	1655.289045	111952.284517	1960.039988	82913.639269
4	0.335579	0.088129	0.143289	0.001701	1630.656199	79667.267654	1948.503884	60204.020268

Let's see if there any missing values:

```
In [98]: useful_data.isna().sum().sum()
Out[98]: 0
```

Fortunately there aren't any missing values. Now well define our X (matrix of features) and Y (output).

```
y = useful_data['label']
X = useful_data.drop(['label'], axis = 1)
```

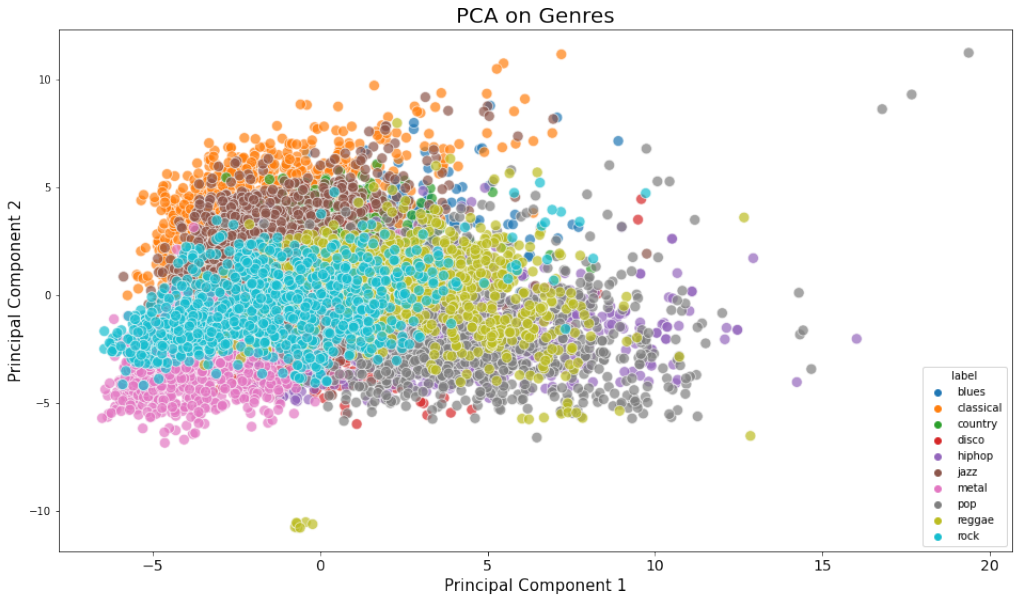
4.2.1 PCA

Let's apply PCA to our data, we'll start by standarizing it (make its mean null and variance equal to one).

```
In [99]: strdScaler = skp.StandardScaler()
         DataPCA = strdScaler.fit_transform(X)
```

```
In [100]: DataPCA
Out[100]: array([[-0.46702077,  0.62722023, -0.03580335, ..., -0.51892536,
                   0.14265042, -0.2935083 ],
                 [-0.38225693,  0.11572084, -0.29355257, ...,  1.01770177,
                   1.27650979,  0.04962287],
                 [-0.34076143,  0.75187636, -0.01253548, ..., -0.04802785,
                   0.66195693, -0.5198386 ],
                 ...,
                 [-0.69698407,  0.63050737,  0.01612527, ..., -0.59584914,
                   1.11438033, -0.25442747],
                 [-0.2589466 ,  0.20664885, -0.79471739, ...,  0.85312722,
                   0.40925854,  0.55015097],
                 [-0.13627844, -0.55660561, -0.57181464, ...,  0.02671313,
                   0.37380199, -0.47169378]])
```

After that, we will project the data on the first two principal components:



The cumulative explained variance isn't high enough (equal to 0.33), the threshold is 0.80, therefore we can't rely only on the the first two principal components.

```
In [42]: sum(pca.explained_variance_ratio_)
Out[42]: 0.3359769718422832
```

4.3 Training the models

4.3.1 Spliting data into test and train sets

In the dataset, there are approximately a 1000 sound clip per genre. in this project we will take a percentage of 80% of the data for training and the rest for testing.
A challenge I faced whilest splitting the data is to have the same number of observations (sound clips) from each genre. A simple solution for this is the following program:

```
In [51]: cutoff = 800
         labels = useful_data['label'].unique()
         trainDF = pd.DataFrame()
         testDF = pd.DataFrame()
         for label in labels:
             dfoflabel = useful_data[useful_data['label'] == label]
             nrows = dfoflabel.shape[0]
             #appending the first 800 sample of each label to train set
             trainDF = trainDF.append(dfoflabel.head(cutoff))
             #appending the rest to test set
             testDF =testDF.append(dfoflabel.tail(nrows - cutoff))
```

here we take the first 800 rows from each genre and append it the *trainDF* and the rest to *testDF*.
That way we make sure that the number of observations in each genre in the training data is the same (800 rows).
After that we extract our *Xtrain* , *Ytrain* , *Xtest* and *Ytest*

```
In [170]: df_Y_train = trainDF.pop('label')
          df_X_train = trainDF
```

```
In [171]: df_Y_test = testDF.pop('label')
          df_X_test = testDF
```

```
In [172]: from sklearn.linear_model import LogisticRegression
          from sklearn.metrics import accuracy_score
```

```
In [173]: sts = skp.StandardScaler()
          le = skp.LabelEncoder()

          X = sts.fit_transform(df_X_train)
          Y = le.fit_transform(df_Y_train)
          Xtest = sts.fit_transform(df_X_test)
          Ytest = le.fit_transform(df_Y_test)
```

4.3.2 Logistic Regression

I tried several classification methods to compare the accuracy for each method: We'll start with logistic regression:

Logistic regression

```
In [178]: from sklearn.linear_model import LogisticRegression

In [179]: clf = LogisticRegression(random_state=0,max_iter = 10000).fit(X, Y)

In [180]: clf.score(X,Y)
Out[180]: 0.763625

In [181]: clf.score(Xtest, Ytest)
Out[181]: 0.5346733668341709
```

We can see that the accuracy for the testing data is 0.5 which is not bad considering that are 10 classes.
Let's see if we can do better.

4.3.3 KNeighborsClassifier

For this part we'll take k = 3, and tune it later on :

KNeighborsClassifier

```
In [53]: from sklearn.neighbors import KNeighborsClassifier
         neigh = KNeighborsClassifier(n_neighbors=3)
         neigh.fit(X, Y)
Out[53]: KNeighborsClassifier(n_neighbors=3)

In [54]: neigh.score(X,Y)
Out[54]: 0.965875

In [55]: neigh.score(Xtest, Ytest)
Out[55]: 0.4291457286432161
```

We can see that we got a high accuracy for the training data but not for the testing data. This might suggest an overfitting problem.
Let's try and find the best value for k:

Finding the best value for k

In [185]:
```
from sklearn.model_selection import GridSearchCV
```

In [187]:
```
grid_params = {
    'n_neighbors' : [3,5,7,9,11,19],
    'weights' : ['uniform','distance'],
    'metric' : ['euclidean',',manhattan']
}

gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv = 3, n_jobs = -1)
gs_results = gs.fit(X,Y)
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   56.4s
[Parallel(n_jobs=-1)]: Done  72 out of  72 | elapsed:   56.8s finished
```

In [194]:
```
print("the best score is {} and the best parameters are {}".format(gs_results.best_score_, gs_results.best_params_)
```

```
the best score is 0.5032469684725268 and the best parameters are {'metric': 'euclidean', 'n_neighbors': 9, 'weight
s': 'distance'}
```

we can see that the best we can do with KNeighbotsClassifier is an accuracy of 50%.

4.3.4 Deep Neural Networks Classifier

In this part, we'll test two neural networks. The first has 2 hidden layers: the first has 128 nodes, the third has 64 nodes.

In [229]:
```
def baseline_model2():
    # create model
    model = Sequential()
    model.add(Dense(128, input_dim=X.shape[1],activation='relu'))
    model.add(Dense(64, activation='relu'))


    model.add(Dense(10, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

In [230]:
```
estimator2 = KerasClassifier(build_fn=baseline_model2, verbose=0)
```

In [233]:
```
batch_size = [10, 20, 40, 60, 80, 100]
epochs = [10, 50, 100]
param_grid = dict(batch_size=batch_size, epochs=epochs)
grid2 = GridSearchCV(estimator=estimator2, param_grid=param_grid, n_jobs=-1, cv=3)
grid_result2 = grid2.fit(X, dummy_y)
print("Best: %f using %s" % (grid_result2.best_score_, grid_result2.best_params_))
```

```
Best: 0.116247 using {'batch_size': 10, 'epochs': 10}
```

In [234]:
```
kfold = KFold(n_splits=10, shuffle=True)
results2 = cross_val_score(grid_result2.best_estimator_, X, dummy_y, cv=kfold)
print("Average Accuracy and its std: %.2f%% (%.2f%%)" % (results2.mean()*100, results2.std()*100))
```

```
Average Accuracy and its std: 88.31% (1.63%)
```

We can see this neural has a good score of 88 percentage with the best parameters being batch size: 10, epochs: 10

the second neural network has 3 hidden layers:
The first has 256 nodes , the second has 128 nodes, the third has 64 nodes.

```
In [203]: from keras.models import Sequential
          from keras.layers import Dense
          from keras.wrappers.scikit_learn import KerasClassifier
          from keras.utils import np_utils
          from sklearn.model_selection import cross_val_score
          from sklearn.model_selection import KFold
          from sklearn.preprocessing import LabelEncoder
          from sklearn.pipeline import Pipeline
```

```
In [211]: def baseline_model():
              # create model
              model = Sequential()
              model.add(Dense(256, input_dim=X.shape[1], activation='relu'))
              model.add(Dense(128, activation='relu'))
              model.add(Dense(64, activation='relu'))


              model.add(Dense(10, activation='softmax'))
              # Compile model
              model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
              return model
```

```
In [223]: dummy_y = np_utils.to_categorical(Y)
          estimator = KerasClassifier(build_fn=baseline_model, verbose=0)
```

Let's find the best values for the epochs and batch size:

```
In [225]: batch_size = [10, 20, 40, 60, 80, 100]
          epochs = [10, 50, 100]
          param_grid = dict(batch_size=batch_size, epochs=epochs)
          grid = GridSearchCV(estimator=estimator, param_grid=param_grid, n_jobs=-1, cv=3)
          grid_result = grid.fit(X, dummy_y)
          print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))

          Best: 0.115997 using {'batch_size': 20, 'epochs': 10}
```

```
In [226]: grid_result.best_estimator_.score(X,dummy_y)

Out[226]: 0.9745000004768372
```

```
In [227]: kfold = KFold(n_splits=10, shuffle=True)
          results = cross_val_score(grid_result.best_estimator_, X, dummy_y, cv=kfold)
          print("Average Accuracy and its std: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))

          Average Accuracy and its std: 89.00% (1.16%)
```

We can see that this neural network has an excellent score of 89.00% with the best parameters being batch size : 20 and epochs : 10

4.3.5 Saving the best model

After training the models and finding the best one, it is time to save it for reuse:

Saving the model

```
In [ ]: grid_result.best_estimator_.model.save("model.h5")
        print("Saved model to disk")
```

5 Conclusion

The deep neural networks method has proven to be the most adequate method for the music genre classification problem.
Now we can easily extract features from a given song and predict its genre via our saved model with an accuracy of 89%.