Music Genre Classification Using Machine Learning

Brahim Erraji

Supervisors : Professeurs Ikram Chairi And Hasnae Zerouaoui

February 7, 2021

Table of contents

Introduction

Nowadays music is everywhere, therefore having a good music classifier is crucial. The applications of music classification are endless:

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

Dataset

The dataset that will be used in this project is the GTZAN dataset.

- over 9990 sound clips with the same duration 30 seconds
- each clip has a label (genre)

which makes it a supervised learning problem.

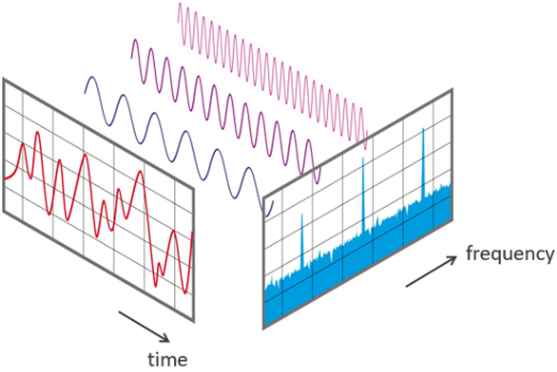
	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var	spectral_bandwidth_mea
0	blues.00000.0.wav	66149	0.335406	0.091048	0.130405	0.003521	1773.065032	167541.630869	1972.74438
1	blues.00000.1.wav	66149	0.343065	0.086147	0.112699	0.001450	1816.693777	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

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
- ..

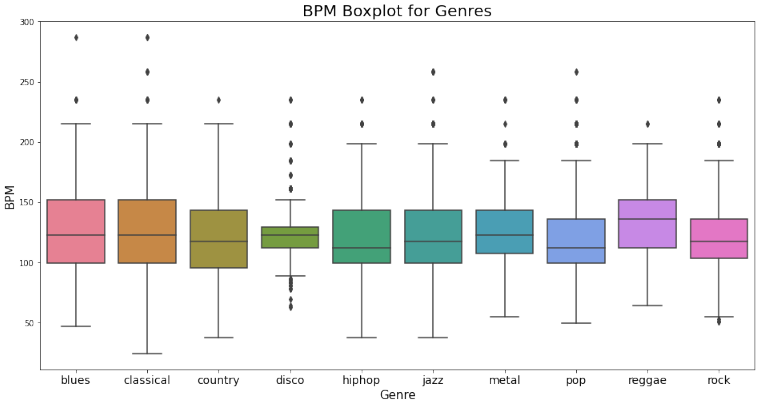
Sound Visualization

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:



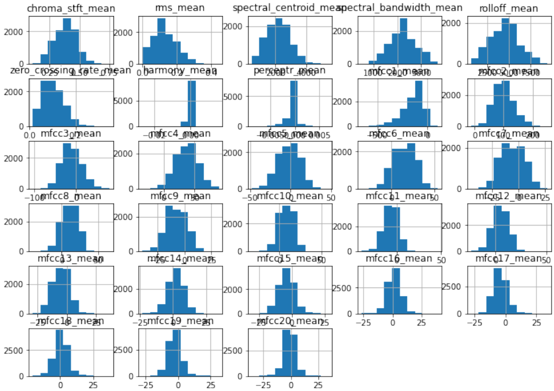
Boxplot of BPM

Below we can see the distribution of the Tempo in each genre:



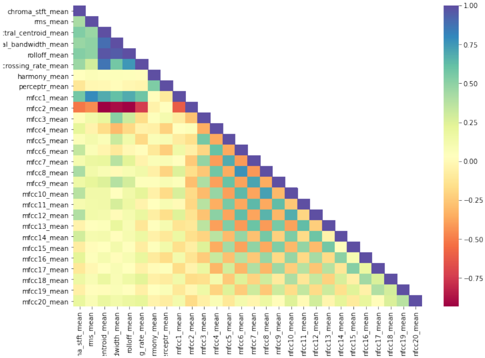
Histogram of each varible

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



Correlation Matrix

One more useful plot is the correlation matrix:



Librosa library in python is used the feature extraction from audio.

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
```

Dataset has two columns that are irrelevant to the genre of the music : *filename* and *length*. Therefore they should be dropped:

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

Missing values

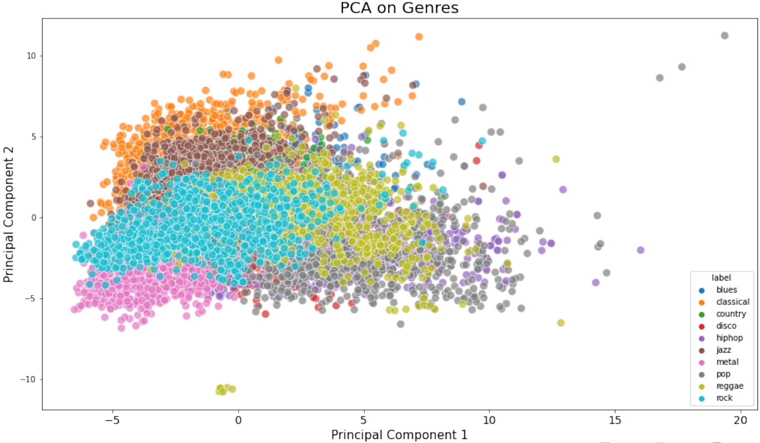
Let's see if there are any missing values:

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

Fortunately, there aren't any missing values

PCA projection

Let's project our data on the first two principal components:



Explained variance

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
```

Splitting the data

A challenge I faced whilest splitting the data is to have the same number of observations (sound clips) from each genre in the training set. 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))
```

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)
```

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
```

- Accuracy for training data : 76%
- Accuracy for test data : 50%

KNeighborsClassifier

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'}
```

- Accuracy for training data : 98%
- Accuracy for test data : 50%

Deep Neural Networks Classifier 1

Two neural networks, First one with 2 hidden layers:
the first has 128 nodes, the second has 64 nodes.

```
In [229]: def baseline_mode run cell, select below
              # 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%)
```

- Accuracy for test data : 88%

Deep Neural Networks Classifier 2

Second one with 3 hidden layers:
the first has 256 nodes, the second has 128 nodes and the third has 64 nodes.

```
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
```

Grid Search to find the best values for batch size and epochs:

```
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%)
```

- Accuracy for test data : 89%

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%.