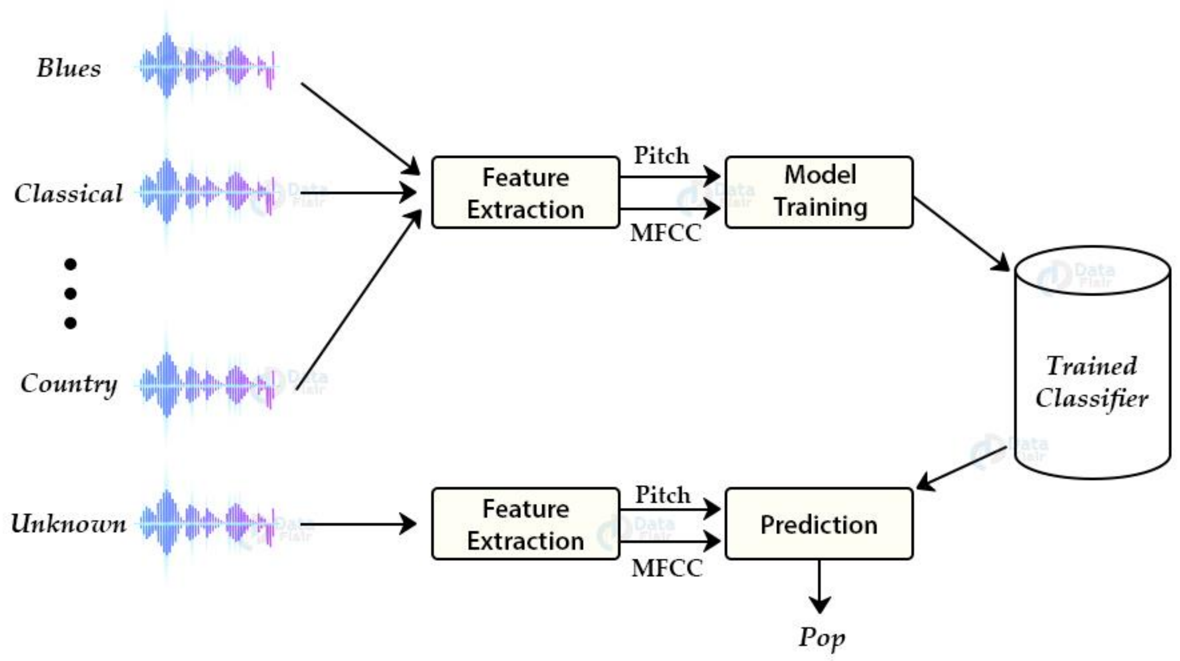
Python Project – Music Genre Classification

Music Genre Classification – Automatically classify different musical genres
In this tutorial we are going to develop a deep learning project to automatically classify different musical genres from audio files. We will classify these audio files using their low-level features of frequency and time domain.

For this project we need a dataset of audio tracks having similar size and similar frequency range. GTZAN genre classification dataset is the most recommended dataset for the music genre classification project and it was collected for this task only.



Music Genre Classification

About the dataset:

The GTZAN genre collection dataset was collected in 2000-2001. It consists of 1000 audio files each having 30 seconds duration. There are 10 classes (10 music genres) each containing 100 audio tracks. Each track is in .wav format. It contains audio files of the following 10 genres:

- Blues
- Classical
- Country
- Disco
- Hiphop
- Jazz
- Metal
- Pop
- Reggae
- Rock

Music Genre Classification approach:

There are various methods to perform classification on this dataset. Some of these approaches are:

- Multiclass support vector machines
- K-means clustering
- K-nearest neighbors
- Convolutional neural networks

We will use K-nearest neighbors algorithm because in various researches it has shown the best results for this problem.

K-Nearest Neighbors is a popular machine learning algorithm for regression and classification. It makes predictions on data points based on their similarity measures i.e distance between them.

Feature Extraction:

The first step for music genre classification project would be to extract features and components from the audio files. It includes identifying the linguistic content and discarding noise.

Mel Frequency Cepstral Coefficients:

These are state-of-the-art features used in automatic speech and speech recognition studies. There are a set of steps for generation of these features:

- Since the audio signals are constantly changing, first we divide these signals into smaller frames. Each frame is around 20-40 ms long
- Then we try to identify different frequencies present in each frame
- Now, separate linguistic frequencies from the noise
- To discard the noise, it then takes discrete cosine transform (DCT) of these frequencies. Using DCT we keep only a specific sequence of frequencies that have a high probability of information.

Steps to build Music Genre Classification:

Download the GTZAN dataset from the following link:

GTZAN dataset

Create a new python file "music_genre.py" and paste the code described in the steps below:

1. Imports:

```
from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np
from tempfile import TemporaryFile
import os
import pickle
import random
import operator
import math
import numpy as np
```

2. Define a function to get the distance between feature vectors and find neighbors:

```
def getNeighbors(trainingSet, instance, k):
distances = []
for x in range (len(trainingSet)):
dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)
distances.append((trainingSet[x][2], dist))
distances.sort(key=operator.itemgetter(1))
neighbors = []
for x in range(k):
neighbors.append(distances[x][0])
return neighbors
```

3. Identify the nearest neighbors:

```
def nearestClass(neighbors):
classVote = {}
for x in range(len(neighbors)):
response = neighbors[x]
if response in classVote:
classVote[response]+=1
else:
classVote[response]=1
sorter = sorted(classVote.items(), key = operator.itemgetter(1), reverse=True)
return sorter[0][0]
```

4. Define a function for model evaluation:

```
def getAccuracy(testSet, predictions):
correct = 0
for x in range (len(testSet)):
if testSet[x][-1]==predictions[x]:
correct+=1
return 1.0*correct/len(testSet)
```

5. Extract features from the dataset and dump these features into a binary .dat file “my.dat”:

```
directory = "__path_to_dataset__"
f= open("my.dat" ,'wb')
```

```
i=0
for folder in os.listdir(directory):
i+=1
if i==11 :
break
for file in os.listdir(directory+folder):
(rate,sig) = wav.read(directory+folder+"/"+file)
mfcc_feat = mfcc(sig,rate ,winlen=0.020, appendEnergy = False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature = (mean_matrix , covariance , i)
pickle.dump(feature , f)
f.close()
```

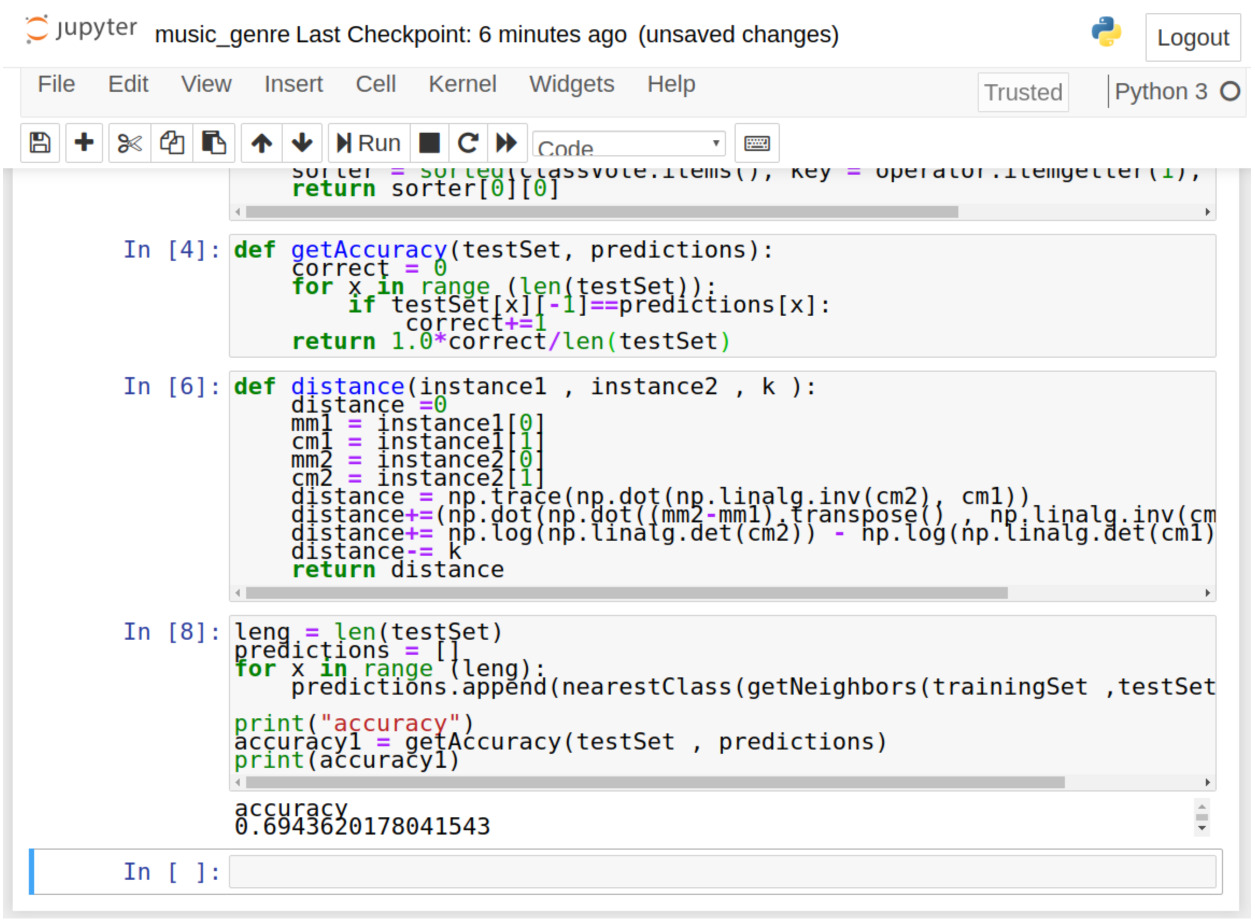
6. Train and test split on the dataset:

```
dataset = []
def loadDataset(filename , split , trSet , teSet):
with open("my.dat" , 'rb') as f:
while True:
try:
dataset.append(pickle.load(f))
except EOFError:
f.close()
break
for x in range(len(dataset)):
if random.random() <split :
trSet.append(dataset[x])
else:
teSet.append(dataset[x])
trainingSet = []
testSet = []
loadDataset("my.dat" , 0.66, trainingSet, testSet)
```

7. Make prediction using KNN and get the accuracy on test data:

```
leng = len(testSet)
predictions = []
for x in range (leng):
predictions.append(nearestClass(getNeighbors(trainingSet ,testSet[x] , 5)))
accuracy1 = getAccuracy(testSet , predictions)
```

print(accuracy1)



```
        sorter = sorted(classVote.items(), key = operator.itemgetter(1),
        return sorter[0][0]
```

```
In [4]: def getAccuracy(testSet, predictions):
            correct = 0
            for x in range (len(testSet)):
                if testSet[x][-1]==predictions[x]:
                    correct+=1
            return 1.0*correct/len(testSet)
```

```
In [6]: def distance(instance1 , instance2 , k ):
            distance =0
            mm1 = instance1[0]
            cm1 = instance1[1]
            mm2 = instance2[0]
            cm2 = instance2[1]
            distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))
            distance+=(np.dot(np.dot((mm2-mm1).transpose() , np.linalg.inv(cm
            distance+= np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1)
            distance-= k
            return distance
```

```
In [8]: leng = len(testSet)
        predictions = []
        for x in range (leng):
            predictions.append(nearestClass(getNeighbors(trainingSet ,testSet

        print("accuracy")
        accuracy1 = getAccuracy(testSet , predictions)
        print(accuracy1)

        accuracy
        0.6943620178041543

In [ ]:
```

Test the classifier with new audio file

Save the new audio file in the present directory. Make a new file test.py and paste the below script:

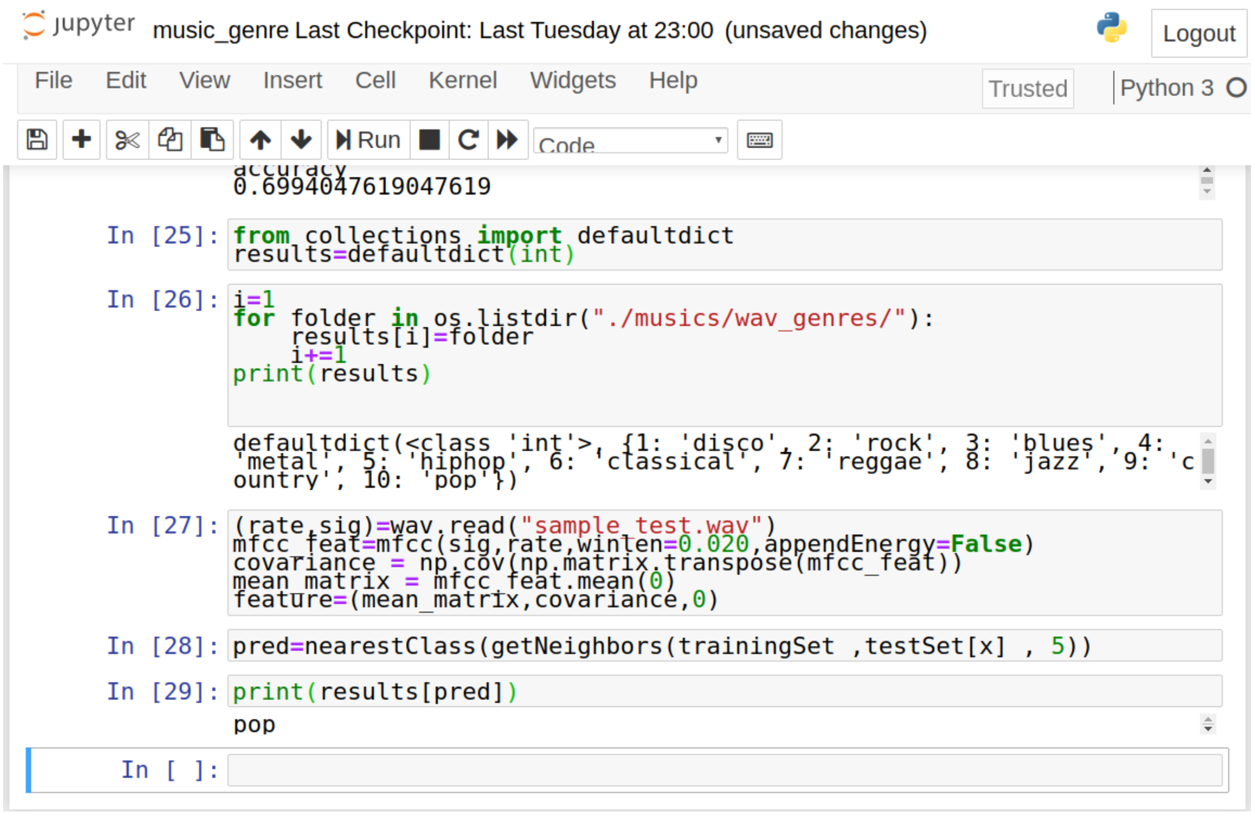
```
from python_speech_features import mfcc
import scipy.io.wavfile as wav
import numpy as np
from tempfile import TemporaryFile
import os
import pickle
import random
import operator
import math
import numpy as np
```

```
from collections import defaultdict
dataset = []
def loadDataset(filename):
with open("my.dat" , 'rb') as f:
while True:
try:
dataset.append(pickle.load(f))
except EOFError:
f.close()
break
loadDataset("my.dat")
def distance(instance1 , instance2 , k ):
distance =0
mm1 = instance1[0]
cm1 = instance1[1]
mm2 = instance2[0]
cm2 = instance2[1]
distance = np.trace(np.dot(np.linalg.inv(cm2), cm1))
distance+=(np.dot(np.dot((mm2-mm1).transpose() , np.linalg.inv(cm2)) , mm2-mm1 ))
distance+= np.log(np.linalg.det(cm2)) - np.log(np.linalg.det(cm1))
distance-= k
return distance
def getNeighbors(trainingSet , instance , k):
distances =[]
for x in range (len(trainingSet)):
dist = distance(trainingSet[x], instance, k )+ distance(instance, trainingSet[x], k)
distances.append((trainingSet[x][2], dist))
distances.sort(key=operator.itemgetter(1))
neighbors = []
for x in range(k):
neighbors.append(distances[x][0])
return neighbors
def nearestClass(neighbors):
classVote ={}
for x in range(len(neighbors)):
response = neighbors[x]
if response in classVote:
classVote[response]+=1
else:
classVote[response]=1
```

```
sorter = sorted(classVote.items(), key = operator.itemgetter(1), reverse=True)
return sorter[0][0]
results=defaultdict(int)
i=1
for folder in os.listdir("./musics/wav_genres/"):
results[i]=folder
i+=1
(rate,sig)=wav.read("__path_to_new_audio_file_")
mfcc_feat=mfcc(sig,rate,winlen=0.020,appendEnergy=False)
covariance = np.cov(np.matrix.transpose(mfcc_feat))
mean_matrix = mfcc_feat.mean(0)
feature=(mean_matrix,covariance,0)
pred=nearestClass(getNeighbors(dataset ,feature , 5))
print(results[pred])
```

Now, run this script to get the prediction:

```
python3 test.py
```


jupyter music_genre Last Checkpoint: Last Tuesday at 23:00 (unsaved changes)

```
accuracy
0.6994047619047619

In [25]: from collections import defaultdict
         results=defaultdict(int)

In [26]: i=1
         for folder in os.listdir("./musics/wav_genres/"):
             results[i]=folder
             i+=1
         print(results)

         defaultdict(<class 'int'>, {1: 'disco', 2: 'rock', 3: 'blues', 4:
         'metal', 5: 'hiphop', 6: 'classical', 7: 'reggae', 8: 'jazz', 9: 'c
         ountry', 10: 'pop'})

In [27]: (rate,sig)=wav.read("sample_test.wav")
         mfcc_feat=mfcc(sig,rate,winlen=0.020,appendEnergy=False)
         covariance = np.cov(np.matrix.transpose(mfcc_feat))
         mean_matrix = mfcc_feat.mean(0)
         feature=(mean_matrix,covariance,0)

In [28]: pred=nearestClass(getNeighbors(trainingSet ,testSet[x] , 5))

In [29]: print(results[pred])

         pop

In [ ]:
```

Summary:

In this music genre classification project, we have developed a classifier on audio files to predict its genre. We work through this project on GTZAN music genre classification dataset. This tutorial explains how to extract important features from audio files. In this deep learning project we have implemented a K nearest neighbor using a count of K as 5.