Music genre classification using machine learning techniques

Yash Raj Kesarwani (2019178), Vasukumar Kotadiya(2019171)

A python-based music genre classification for the required music(.wav) file music machine learning paradigm. 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.

How does it work?

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.

File Structure

📦Music Recognition

├ 📂genres

| ├ 📂blues

| ├ 📂classical

| ├ 📂country

Modules Required

1. Tensorflow
2. python_speech_features
3. scipy
4. numpy

music_genre.py

This file contains code for reading and loading the training data into my.dat file. It also consists of logic necessary to implement the steps required for classification of music such as getting accuracy, make prediction using KNN and get the accuracy on test data, building and extracting features from the dataset and dump these features into a binary .dat file “my.dat”.

testing .py

This file is used to test and classify music genre of the specified file (which is specified within the code).

Dataset used:

We have used 1-benchmark datasets to evaluate the model. The details of the datasets used & the summary of results are attached below.

GTZAN dataset:

The dataset consists of 1000 audio tracks each 30 seconds long. It contains 10 genres, each represented by 100 tracks. The tracks are all 22050Hz Mono 16-bit audio files in .wav format. https://www.tensorflow.org/datasets/catalog/gtzan

Run Code:

1. Download GTZAN music classification data and run music_genre.py file to train dataset and create my.dat file which contains result of training.("my.dat" has already been created).
2. Specify music file inside testing.py file and run testing.py to get genre of music.

Possible changes:

There can be various other methods for using given dataset to classify music according to genre like K-means clustering method instead of K-nearest clustering.

Code:

```
#music_genre.py
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

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]


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)


directory = "genres/"
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()


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)


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

accuracy1 = getAccuracy(testSet , predictions)
print(accuracy1)
```

```
# Testing test.wav
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("./genres/"):
    results[i]=folder
    i+=1


(rate,sig)=wav.read("example.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)


pred=nearestClass(getNeighbors(dataset ,feature , 5))


print(results[pred])
```