import numpy

import librosa

import sklearn

import joblib

import config

def main():

path = librosa.util.find\_files(config.Test.TEST\_DATA\_PATH)

sample1 = config.CreateDataset.SAMPLING\_RATE

hsize = config.CreateDataset.HOP\_SIZE

frames = config.CreateDataset.FRAME\_SIZE

music = []

print("Extracting sample arrays for files...")

for p in path:

x, sr = librosa.load(p, sr=sample1, duration=5.0)

music.append(x)

print("DONE!")

print("Extracting features from sample arrays...")

data = numpy.array([extract\_features(x, sample1, frames, hsize) for x in songs])

print("DONE!")

scaler = sklearn.preprocessing.MinMaxScaler(feature\_range=(-1, 1))

data = scaler.fit\_transform(data)

# get the model from pkl file

svm = joblib.load('model.pkl')

print("----------------------------------- Predicted Labels -----------------------------------\n")

preds = svm.predict(data)

print(preds)

print("")

print("----------------------------------------------------------------------------------------")

def extract\_features(signal, sample1, frames, hsize):

zero\_crossing\_rate = librosa.feature.zero\_crossing\_rate(y=signal, frame\_length=frames, hop\_length=hsize)

spectral\_centroid = librosa.feature.spectral\_centroid(y=signal, sr=sample1, n\_fft=frames,

hop\_length=hsize)

spectral\_contrast = librosa.feature.spectral\_contrast(y=signal, sr=sample1, n\_fft=frames,

hop\_length=hsize)

spectral\_bandwidth = librosa.feature.spectral\_bandwidth(y=signal, sr=sample1, n\_fft=frames,

hop\_length=hsize)

spectral\_rolloff = librosa.feature.spectral\_rolloff(y=signal, sr=sample1, n\_fft=frames,

hop\_length=hsize)

mfccs = librosa.feature.mfcc(y=signal, sr=sample1, n\_fft=frames, hop\_length=hsize)

return [numpy.mean(zero\_crossing\_rate),

numpy.std(zero\_crossing\_rate),

numpy.mean(spectral\_centroid),

numpy.std(spectral\_centroid),

numpy.mean(spectral\_contrast),

numpy.std(spectral\_contrast),

numpy.mean(spectral\_bandwidth),

numpy.std(spectral\_bandwidth),

numpy.mean(spectral\_rolloff),

numpy.std(spectral\_rolloff),

numpy.mean(mfccs[1, :]),

numpy.std(mfccs[1, :]),

numpy.mean(mfccs[2, :]),

numpy.std(mfccs[2, :]),

numpy.mean(mfccs[3, :]),

numpy.std(mfccs[3, :]),

numpy.mean(mfccs[4, :]),

numpy.std(mfccs[4, :]),

numpy.mean(mfccs[5, :]),

numpy.std(mfccs[5, :]),

numpy.mean(mfccs[6, :]),

numpy.std(mfccs[6, :]),

numpy.mean(mfccs[7, :]),

numpy.std(mfccs[7, :]),

numpy.mean(mfccs[8, :]),

numpy.std(mfccs[8, :]),

numpy.mean(mfccs[9, :]),

numpy.std(mfccs[9, :]),

numpy.mean(mfccs[10, :]),

numpy.std(mfccs[10, :]),

numpy.mean(mfccs[11, :]),

numpy.std(mfccs[11, :]),

numpy.mean(mfccs[12, :]),

numpy.std(mfccs[12, :]),

numpy.mean(mfccs[13, :]),

numpy.std(mfccs[13, :]),

]

if \_\_name\_\_ == '\_\_main\_\_':

main()