SIT789 – Robotics, Computer Vision and Speech Processing

Pass Task 9.2: Speaker recognition using GMMs

Objectives

The objectives of this lab include:

- Implementing a speaker recognition algorithm using GMMs
- Practising with MFCC features to characterise speakers' voice.

Tasks

1. Building GMMs with MFCC features

In this task, we will build GMMs with MFCC features for a speaker recognition application. You first download the SpeakerData folder in the supplied resources, then place this folder into your working directory. This speaker dataset is a part of the VoxForge dataset.

In the SpeakerData folder, there are two folders: Train and Test containing training and test speech files respectively. Each Train/Test folder includes twenty-five (25) sub-folders containing speech sounds made by 25 different speakers. There are ten (10) speech files per speaker, in which three (3) files are used for training and the remainders are used for testing. Not any file is used for both training and testing.

The code below performs MFCC feature extraction from audio data and builds GMMs for all the speakers in the SpeakerData dataset using MFCC features. Although this code is provided to you, you are encouraged to read through and understand it, especially the GMM library. You should also visit sklearn.mixture for more details on GMM usage.

```
import numpy as np
import librosa
from pydub import AudioSegment
from pydub.utils import mediainfo
from sklearn import preprocessing

def mfcc_extraction(audio_filename, #.wav filename
                    hop_duration, #hop_length in seconds, e.g., 0.015s (i.e., 15ms)
                    num_mfcc #number of mfcc features
                   ):
    speech = AudioSegment.from_wav(audio_filename) #Read audio data from file
    samples = speech.get_array_of_samples() #samples x(t)
```

```
    sampling_rate = speech.frame_rate #sampling rate f

    mfcc = librosa.feature.mfcc(
        y = np.float32(samples),
        sr = sampling_rate,
        hop_length = int(sampling_rate * hop_duration),
        n_mfcc = num_mfcc)


    return mfcc.T

from sklearn.mixture import GaussianMixture

def learningGMM(features, #list of feature vectors, each feature vector is an array
                n_components, #the number of components
                max_iter #maximum number of iterations
               ):
    gmm = GaussianMixture(n_components = n_components, max_iter = max_iter)
    gmm.fit(features)
    return gmm
```

To build GMMs for speakers, we need to define speakers and load their training data. Since each speaker has a folder with their name in the Train/Test folder, the list of speakers can be loaded from the list of sub-folders in the Train/Test folder as follows.

```
import os
path = 'SpeakerData/'
speakers = os.listdir(path + 'Train/')
print(speakers)
```

Now, we load the training data for each speaker and extract the MFCC features from the training data.

```
from sklearn import preprocessing

mfcc_all_speakers = [] #list of the MFCC features of the training data of all speakers
hop_duration = 0.015 #15ms
num_mfcc = 12

for s in speakers:
    sub_path = path + 'Train/' + s + '/'
    sub_file_names = [os.path.join(sub_path, f) for f in os.listdir(sub_path)]
    mfcc_one_speaker = np.asarray(())
    for fn in sub_file_names:
        mfcc_one_file = mfcc_extraction(fn, hop_duration, num_mfcc)
        if mfcc_one_speaker.size == 0:
            mfcc_one_speaker = mfcc_one_file
        else:
            mfcc_one_speaker = np.vstack((mfcc_one_speaker, mfcc_one_file))
    mfcc_all_speakers.append(mfcc_one_speaker)
```

As feature extraction is time consuming, we should save the features to files; each file stores the MFCC features extracted from the speech data of one speaker. Suppose that the features of the training data of all speakers are stored in a folder named TrainingFeatures (you need to create this folder if it does not exist).

```
import pickle

for i in range(0, len(speakers)):
    with open('TrainingFeatures/' + speakers[i] + '_mfcc.fea','wb') as f:
        pickle.dump(mfcc_all_speakers[i], f)
```

We now build our GMMs using the following code.

```
n_components = 5
max_iter = 50

gmms = [] #list of GMMs, each is for a speaker
for i in range(0, len(speakers)):
    gmm = learningGMM(mfcc_all_speakers[i],
                n_components,
                max_iter)
    gmms.append(gmm)
```

We also save the GMMs to files. Suppose the GMM for each speaker is saved in one file named by the speaker's name and with extension '.gmm', and all GMMs of all speakers are stored in a folder named Models (you need to create this folder if it does not exist).

```
for i in range(len(speakers)):
    with open('Models/' + speakers[i] + '.gmm', 'wb') as f: #'wb' is for binary write
        pickle.dump(gmms[i], f)
```

2. Speaker recognition using GMMs

In this section, we will use the trained GMMs to build a speaker recognition algorithm. We first load the GMMs from files using the following code.

```
gmms = []
for i in range(len(speakers)):
    with open('Models/' + speakers[i] + '.gmm', 'rb') as f: #'wb' is for binary write
        gmm = pickle.load(f)
        gmms.append(gmm)
```

You are required to implement a speaker recognition method named speaker_recognition. This method receives input as a speech file name and a list of GMMs, and returns the ID of the speaker who most likely made the input speech. Specifically, the speaker_recognition method should follow the template:

```
def speaker_recognition(audio_file_name, gmms):
    speaker_id = 0 #you need to calculate this
    return speaker_id
```

Hint: the likelihood that a given speech is made by a speaker i is computed as gmms[i].score(f) where f is the MFCC features extracted using the mfcc_extraction method. The greater gmms[i].score(f) is, the more likely speaker i is the speaker of the speech. The speaker k who has the highest gmms[k].score(f) is considered as the one who most likely made the input speech.

To identify the speaker of a given a speech sound, e.g., SpeakerData/Test/Ara/a0522.wav, we perform:

```
speaker_id = speaker_recognition('SpeakerData/Test/Ara/a0522.wav', gmms)
print(speakers[speaker_id])
```

You then test the speaker recognition algorithm on the entire test set and report the overall recognition accuracy and confusion matrix (see Task 4.1 and Task 8.1 for evaluation of algorithms).

Further practice (optional)
You can also test the algorithm with your own data. For instance, you can consider yourself a speaker and create a folder with your name in both SpeakerData/Train and SpeakerData/Test to store training/test recordings of your voice. Following the setting used in SpeakerData, you should record 10 utterances (around 1 second per utterance), then use 3 utterances for training and 7 utterances for testing. Note that your recordings should not contain pause frames (i.e., unvoiced frames). Since this task is speaker recognition (not speech recognition), the utterances can be spoken in any languages.

Submission instructions

1. Perform tasks required in Section 1 and 2.
2. Complete the supplied answer sheet.
3. Submit your code (.py) and answer sheet (.pdf) to OnTrack.