Untitled28

September 10, 2023

1 SIT789 - Applications of Computer Vision and Speech Processing

1.1 Credit Task 9.2: Speaker recognition using GMMs

1. Building GMMs with MFCC features

[1]:
```
import numpy as np
import librosa
from pydub import AudioSegment
from pydub.utils import mediainfo
from sklearn import preprocessing
import glob

import warnings

warnings.filterwarnings("ignore", category=UserWarning)
```

```
C:\Users\vinit\anaconda3\lib\site-packages\pydub\utils.py:170: RuntimeWarning:
Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not work
  warn("Couldn't find ffmpeg or avconv - defaulting to ffmpeg, but may not
work", RuntimeWarning)
```

[2]:
```
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
    )  # Compute MFCC features
```

```
    return mfcc.T
```

[3]:
```
from sklearn.mixture import GaussianMixture

def learningGMM(features,
                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.

[4]:
```
import os
path = 'SpeakerData/'
speakers = os.listdir(path + 'Train/')
print(speakers)
```

```
['Anthony', 'AppleEater', 'Ara', 'Argail', 'Ariyan', 'Arjuan', 'Artem',
'Arthur', 'Artk', 'Arun', 'Arvala', 'Asalkeld', 'Asladic', 'Asp', 'Azmisov',
'B', 'Bachroxx', 'Bae', 'Bahoke', 'Bareford', 'Bart', 'Bassel', 'Beady', 'Beez',
'BelmontGuy']
```

[5]:
```
from sklearn import preprocessing

# This list is used to store the MFCC features of all training data of all␣
 ↪speakers
mfcc_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 all the features are stored in a folder named TrainingFeatures.

[6]:
```
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

[7]:
```
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. All the GMMs are stored in a folder named Models, the GMM for each speaker is saved in one file.

[8]:
```
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 We first load the GMMs from files using the following code.

[9]:
```
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)
```

Loading the MFCC features

[10]:
```
mfcc_features = []

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

```
        mfcc = pickle.load(f)
        mfcc_features.append(mfcc)
```

[11]:
```
hop_duration = 0.015
num_mfcc = 12

def speaker_recognition(audio_file_name, gmms):
    scores = []

    for i in range(len(gmms)):
        f = mfcc_extraction(audio_file_name, hop_duration, num_mfcc)

        scores.append(gmms[i].score(f))
    speaker_id = scores.index(max(scores))

    return speaker_id
```

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

[12]:
```
speaker_id = speaker_recognition('SpeakerData/Test/Ara/a0522.wav', gmms)
print(speakers[speaker_id])
```

```
Ara
```

Test the algorithm on the entire test set and report the recognition accuracy

[13]:
```
pred_labels = []
true_labels = []

for folder_name in sorted(glob.glob('SpeakerData/Test/*')):
    for file_name in sorted(glob.glob(folder_name+"/*")):
        speaker_id = speaker_recognition(file_name, gmms)
        predicted_label = speakers[speaker_id]
        true_label = folder_name.split('/')[-1]
        pred_labels.append(predicted_label)

        true_labels.append(true_label)

true_labels = [e[5:] for e in true_labels]
```

[14]:
```
from sklearn.metrics import classification_report, confusion_matrix ,␣
 ↪accuracy_score

cm = confusion_matrix(true_labels, pred_labels)

print (cm)
```

```
[[1 0 0 0 0 0 0 6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```
 [0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 0 0 4 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0 0 0]
 [0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 7]]
```

[15]:

```python
print(classification_report(true_labels, pred_labels, labels=speakers))

print("Overall Recognition Accuracy: {}".format(accuracy_score(true_labels,
 ↪pred_labels)))
```

```
              precision    recall  f1-score   support

     Anthony       1.00      0.14      0.25         7
  AppleEater       1.00      1.00      1.00         7
         Ara       1.00      1.00      1.00         7
      Argail       1.00      1.00      1.00         7
      Ariyan       1.00      1.00      1.00         7
      Arjuan       1.00      1.00      1.00         7
       Artem       1.00      1.00      1.00         7
      Arthur       0.44      1.00      0.61         7
        Artk       1.00      1.00      1.00         7
        Arun       1.00      1.00      1.00         7
      Arvala       1.00      1.00      1.00         7
    Asalkeld       1.00      1.00      1.00         7
     Asladic       1.00      1.00      1.00         7
         Asp       1.00      1.00      1.00         7
     Azmisov       1.00      1.00      1.00         7
           B       1.00      1.00      1.00         7
```

```
    Bachroxx       1.00      1.00      1.00         7
         Bae       1.00      1.00      1.00         7
      Bahoke       1.00      0.43      0.60         7
    Bareford       1.00      1.00      1.00         7
        Bart       1.00      0.57      0.73         7
      Bassel       0.64      1.00      0.78         7
       Beady       1.00      1.00      1.00         7
        Beez       1.00      1.00      1.00         7
  BelmontGuy       1.00      1.00      1.00         7

    accuracy                           0.93       175
   macro avg       0.96      0.93      0.92       175
weighted avg       0.96      0.93      0.92       175

Overall Recognition Accuracy: 0.9257142857142857
```

Observation

- Upon evaluating the algorithm on the complete test dataset, we achieved an Overall Recognition Accuracy of 92.57%, indicating that the model accurately predicted the majority of speakers with a high level of accuracy.