Lab 9 & 10 - Voice Recognition

Classical Voice Recognition

In the classical approach we will use Mel-frequency cepstral coefficients (MFCCs) as features and Gaussian mixture models (GMMs) to classify them.

First of all, download the Voice Dataset:
https://drive.google.com/file/d/1BPIH87RhrypLmsIdrSX5zoPOMezr17yq/view?usp=sharing
It consists of a single word recorded by 9 different people in the .wav format and it's divided into train and test subset. Our goal is to extract MFCC from each train recording and fit GMM to them. After that we will validate our simple system on the test subset.

1. Import all necessary libraries: numpy, os, librosa (for MFCC extraction) and from sklearn.mixture import GaussianMixture.
2. Training process - for each recording from the train subset:
    a. Load recording - librosa.load;
    b. Extract MFCC features - librosa.features.mfcc with `n_mfcc=39` and `sr` set accordingly to the input data (later try manipulate the n_mfcc value and observe the results);
    c. Fit GMM to the extracted features - GaussianMixture with `n_components=32` and `random_state=0` (later try to manipulate the n_components value and observe the results). Note that - according to the sklearn documentation - features' vector should be transposed to form a row vector instead of column vector (you can use transpose from numpy);
    d. Add GMM to some collection (e.g. append to the list).
3. After successful training, classify the test recordings based on GMMs - for each test recording:
    a. Load and extract MFCC;
    b. For each GMM in the collection, compute the score for MFCC extracted from the test recording. GMM with the maximum score represents the detected speaker.
4. Now try to manipulate parameters of MFCC extraction and GMM fitting - write down (e.g. in the comments) the achieved results.

AI-based Voice Recognition (EXTENSION EXERCISE)

Artificial Intelligence can be used in voice recognition tasks in many different ways. This exercise presents the example of an AI-based system to perform so-called “speaker embedding”. Its goal is to extract fixed-length feature vectors from variable-length audio recordings. Their similarity can be further measured using e.g. cosine similarity (like in normal linear algebra, because discussed vectors belong to the same linear space) or PLDA (Probabilistic Linear Discriminant Analysis). Based on the similarity value, we can decide whether the audio files under consideration were recorded by the same person or not.

In our exercise we will use the Densely Connected Time Delay Neural Network (D-TDNN) proposed by Yu & Li in this paper. The official implementation of this work in PyTorch library is available on github: https://github.com/yuyq96/D-TDNN.

1. Clone the repository with D-TDNN. You may feel encouraged to read the readme file.
2. To avoid using an additional program (Kaldi toolkit), checkout the *kaldifeat* branch. Here only a lightweight Kaldifeat toolkit is needed, which is provided also on github repository: https://github.com/yuyq96/kaldifeat. Clone this repo and place it in the folder with D-TDNN (i.e. kaldifeat folder with *feature.py* and *ivector.py* should be placed in the main D-TDNN folder with *evaluate.py*).
3. To run the network we need a pre-trained model, voice recordings for evaluation and two additional files: *trials* (with list of tested recordings pairs - given by the unique labels - and expected outcomes) and *wav.scp* (to show the network where are the wav files given by the labels).
4. Firstly, we will use the VoxCeleb1 dataset, on which D-TDNN was tested by the authors. Download the zip archive with wav files from https://drive.google.com/file/d/1dZH0666ZNPDRG6MYrf0_6H2o45j9U-GO/view?usp=sharing. You can extract the archive to any folder on your computer, but it seems reasonable to create a new folder, e.g. *data_dir* in the D-TDNN directory and extract the archive there. Take a moment to examine the structure of the VoxCeleb dataset. As you can see, in the *wav* directory we have folders representing the different speakers (given by unique id). In each of them there are folders with somehow *strange* names (similar to the part in the link to youtube’s video), which contain multiple voice recordings.
5. Download the trials file and place it in the data_dir directory. As you can see, each line consists of three elements: label of the first recording, label of the second recording and target/nontarget string, which indicates whether they are recordings of

the same person (i.e. “target” case) or not. Note the structure of unique labels: person_id-YT_link-recording_number.

6. The *wav.scp* file we have to prepare ourselves. Each line of it should correspond to exactly one recording in the dataset and has the following structure: *label path*, e.g:

```
id10301-rXRbmL7nzIo-00002
./data_dir/VoxCeleb1_test_wav/wav/id10301/rXRbmL7nzIo/00002.wav
```

   Of course we can do it manually (good luck with 4874 recordings), but it seems that using Python script for this task is the better option. Some hints:
   - labels must be the same as in the *trials* file (e.g. you can use walk function from os, split the path and construct proper labels),
   - paths can be relative, starting from the directory containing *eavaluation.py* script,
   - end the file line with ‘\n’ char.

   Place the prepared *wav.scp* file where you previously placed the *trials* file (*data_dir* directory).
7. Download the pre-trained model and place it in the main D-TDNN folder (where you have *evaluate.py* script).
8. Now we can finally try to run the *evaluate.py* script. Read and understand it - install all necessary packages and set the proper value of *num_workers* in *DataLoader*, depending on your hardware. After that, run script with the following (or similar) commend:

```
python evaluate.py --root data_dir --model D-TDNN --checkpoint
dtdnn.pth --device cpu
```

   Note. You can also try `–device cuda` option, but it may crash due to the problems with CUDA configuration.
   You should see outcomes (write down values in comments) similar to the ones reported by authors (Evaluation section in github Readme, results for D-TDNN). Take a moment to understand how vectors similarity is measured and what metrics are used to check the network performance.
   In case of problems, here is some basic troubleshooting:
   - *BrokenPipeError* - set the *num_worker* to 0;
   - *RuntimeError: Expected object of scalar type Double but got scalar type Float* - change data type before sending to the device (before line 59 in *evaluate.py*

script):

```
data=data.float()
data = data.to(device)
```

9. As a final task, we will use the D-TDNN network for speaker embedding on our own dataset (*voice_samples*), prepared during the classical part. Once again, we have to create proper *trials* and *wav.scp* files. For the first one you can use the following Python script:
   https://drive.google.com/file/d/1hpF3v-Ineb7-nQc7OCfUr3TMJiXJjgHT/view?usp=sharing
   Note how labels are constructed in this case. Based on that, create a wav.scp file and run the *evaluate.py* script. Despite the fact that the network was trained on the VoxCeleb dataset, it allows us to extract meaningful vectors from our own collection of recordings.

After completing the exercise, please upload to the UPEL the *evaluation.py* script with the obtained outcomes in the comments.