CSC2511 Assignment 3 Bonus

Tianxiang Chen 999473181

This document records some ideas I implemented for the bonus part. I made attempts for 3 sections of bonus: *voice banking, dimensionally reduction and ASR with sequence-to-sequence models*.

1. Voice Banking

My username is **TianxiangChen** and register email is tianxiang.chen@mail.utornoto.ca.

I complete 11 sessions at the time when A3 is due. Most of the sessions were finished separated by at least 20 hours, except for the first two. I made my second session immediately after the first one because I am not sure my microphone works or not for the first session.

This work still has some difficulty for me especially for those paragraph memorization and picture description since sometimes I forgot what to say (I am not an English native speaker). I will probably continue to do some more sessions.

One suggestion for this program is adding more data. For example, for the 11 sessions I finished, I have seen some 'paragraph memorization' multiple times, e.g. the Limpy duck, my 93-year-old grandfather and rainbow. When a student did this for 10 days, he/she probably has memorized those paragraphs to some extent, which might violate the original purpose for this program.

2. Dimensionality Reduction

I used the PCA function from sklearn for this part. Notice for the skeleton code given, the program goes through each speaker's folder and randomly pop one .npy file as the test data while keeping the rest as training. To keep consistency of the train-test data combination for different component number d, I divided the data into train and test set at first. Then, when I swept the component number n from 13 to 1, the experiments were using the same train-test set. This approach still follows the 'pop one npy file as test' so it still has the randomness as every time you run the code, the test set would be different.

My approach is firstly initializing a PCA model and de-composite the number of features for the training data to d and build the GMM model based on the reduced PCA train data. Then I fit the test data into the PCA model build before, thus getting a test data set dimensionally reduced to d and measuring the log likelihood as in part 2.3.

The code is named **bonus_pca.py** and the result is saved in **bonus_pca.txt**.

I kept all the parameters as default and swept the dimension d reduced to by PCA. The result could be summarized in the table below.

Dimension	Accuracy
13	1.0
12	1.0
11	1.0
10	1.0
9	1.0
8	1.0
7	1.0
6	1.0
5	1.0
4	0.96875
3	1.0
2	0.9375
1	0.6875

Table 1 Accuracy for PCA reduction

From this table, we can see that PCA works well for this GMM classification problems. With the original dimension as 13, we can reduce it to about half, e.g. 7, and still get a good accuracy. Problems occurs when dimension is reduced to extreme small number, e.g. less than 5, which is reasonable since it ignores too much information in that way.

3. ASR with Sequence-to-Sequence Models

Due to time and resource limit, for this part, I used a pre-trained model from Mozilla. This pre-trained model is fairly large (~2.1GB) so as discussed with TA I won't upload the model. Instead, I will briefly describe how to setup the environment.

The model is available on Mozilla's GitHub page, which is available through:

https://github.com/mozilla/DeepSpeech/releases/download/v0.1.1/deepspeech-0.1.1-models.tar.gz

By extracting it, you can get a folder called **models**, with four files in it: lm.binary, alphabet.txt

output_graph.pb, trie.

My setup requires to put the **models** folder inside a folder called **speech**, and the **speech** is on the same level as the code **bonus_deepspeech.py**.

So, the directory setup for the deepspeech would be like:

```
├── bonus_deepspeech.py
├── speech
│   ├── models
│           └── alphabet.txt
│           └── lm.binary
│           └── output_graph.pb
│           └── tire
```

The code is in **bonus_deepspeech.py**. This code takes a long time for running. The result is in **bonus_deepspeech.txt**. The format for the result file is similar to asrDiscussion.txt for Part3.

The last line summarized the performance of the Google, Kaldi and DeepSpeech (DeepS).

	Mean	Std
DeepS	0.462840	0.145399
Google	0.381709	0.132185
Kaldi	0.252818	0.119997

Table 2 Performance Comparison

From the result, it seems DeepS based on Mozilla has the worse performance compared with Google and Kaldi. I manually checked the DeepS's recognized transcript (I just output it to the terminal, not saving it), the main problem is it sometime recognized a single word separately, e.g. 'becaus e' rather than 'because'. Also, the emotion words are mostly different probably due to the training data I guess ('um' vs 'ah'). Beside these, it also recognized some words completely wrong (even doesn't look like an English word). These are caused from the training data Mozilla used.