CS 753: Assignment #1

Instructor: Preethi Jyothi Email: pjyothi@cse.iitb.ac.in

January 28, 2021

Instructions: This assignment is due on or before 11.55 pm on February 13th, 2021. No grace period will be granted for this assignment. The submission portal on Moodle will be closed after 11.55 pm on February 13.

- This is an individual assignment. You will be building an ASR system for Swahili using the **Kaldi** toolkit.
- You will need to install the latest version of Kaldi available here. You can also install Kaldi using docker. (Instructions for using docker are available here.) **Please install the toolkit early and reach out to the TAs in case of any issues.**
- Click here for a detailed structure of your final submission directory. This assignment will be auto-graded. Hence, it is **very important that you do not deviate from the specified structure.** Deviations from the specified structure will be penalized. All the files that need to be submitted are highlighed in red below; all the submitted files will be within the parent directory `submission/`. Compress your submission directory using the command: `tar -cvzf submission.tgz submission` and upload `submission.tgz` to Moodle.

Automatically Recognizing Swahili Speech

Swahili is a language spoken in Africa written using the Latin alphabet. During the course of this assignment, you will develop an ASR system for Swahili. Download a baseline recipe for Swahili provided here. If Kaldi is installed in the directory `[kaldi]`, untar `assgmt1.tgz` within `[kaldi]/egs` to get a directory `assgmt1/recipe`.

Command Line

```
$ cd [kaldi]/egs/assgmt1/recipe
```

Note Set your working directory to be `[kaldi]/egs/assgmt1/recipe`. All subsequent evaluations will be done on a small test set in `data/test` containing around 1500 words.

`run.sh` within `assgmt1/recipe` is the main wrapper script which you will be able to run in under 5 minutes (on a machine with four cores) at the end of task 0. Go through this script carefully to understand the various steps involved. You can set the variable `stage` to determine which stages of `run.sh` will be processed.

Task 0: Setting up the baseline system [6 points]

This exercise is to get you warmed up to Kaldi's file formats. Within `corpus/data`, you will find `wav/`, `example/`, `data-info.txt` and `transcriptions.txt`. `wav/` contains folders for each speaker (like SWH-05-20110327, etc.) and within each speaker folder, there are audio files each containing an utterance by

that speaker (e.g. `SWH-05-20110327_16k-emission_swahili_05h30_-_06h00_tu_20110327_part126.wav`). Transcriptions for each utterance are in `transcriptions.txt`. (Note how the utterance ID has the speaker ID as a prefix.)

Our data is split into `train`, `test` and `truetest` datasets. `data-info.txt` contains information about which speakers belong to which dataset. Each data split contains all the utterances spoken by each speaker. In this task, you must write a script `data-prep.sh` that creates, within `corpus/data`, three folders named `train`, `test` and `truetest` and 4 files (`text`, `wav.scp`, `spk2utt`, `utt2spk`) within each of these three folders. See the `example/` folder for understanding the file format and relative file paths. (You can also visit this site to understand the format better.) Ensure that all your files are sorted. To do this, set `LC_ALL=C` and run `sort -o file file`. In case your sort doesn't support the -o switch, use `sort -u file > tmp && mv tmp file`.

Format

- Each line in `wav.scp` looks like: `<utt-id> <audio-file-path>`. The `<audio-file-path>` must be relative to the working directory `[kaldi]/egs/assgmt1/recipe`.
- Each line in `text` looks like: `<utt-id> <transcription>`.
- Each line in `utt2spk` is of the form: `<utt-id> <speaker-id>`
- Each line in `spk2utt` is of the form: `<speaker-id> <utt-id1> <utt-id2>`. You can generate `spk2utt` automatically from `utt2spk` by using the following Kaldi utility:

Command Line

```
$ [ ! -L "utils" ] && ln -s ../../wsj/s5/utils
$ ./utils/utt2spk_to_spk2utt.pl <utt2spk-file> > <spk2utt-file>
```

Submit the folders `corpus/data/train`,`corpus/data/test`,`corpus/data/truetest`, along with the script `data-prep.sh` as `task0/data/train`, `task0/data/test`,`task0/data/truetest`, `task0/data-prep.sh`. (Any additional scripts `data-prep.sh` needs should be added to the `task0/` directory.)

Finally, check that the baseline system works by executing:

Command Line

```
$ bash run.sh
```

On running `run.sh`, you may get a WER close to 64.3%. Also note the messages on the command line when you execute `run.sh`. (If you see an error of the type "... data/test/utt2spk has invalid newline", you might need to add a newline at the end of these files to make this error disappear.)

Submit the file `run.sh` with relevant code for all the eight tasks mentioned below.

Task 1: Building improved monophone HMMs [3 points]

The following command within `run.sh` is used to train monophone HMMs for the acoustic model:

Command Line

```
$ steps/train_mono.sh --nj 4 --cmd "$train_cmd" \
        data/train data/lang exp/mono
```

Question 1

Within `steps/train_mono.sh`, look at the variables listed on lines 11–30 (within comments "Begin configuration section" and "End configuration section"). Figure out which variables are important by tuning on your test set. Submit a text file `task1/wer.txt` that only contains the best WER on `data/test` you obtained using your final tuned hyperparameters. Also, submit a new `task1/train_mono.sh` with updated hyperparameters that we will use to train monophone HMMs and recover your reported number in `task1/wer.txt`.

Task 2: Train tied-state triphone HMMs [3 points]

Uncomment the following lines to train tied-state triphone HMMs.

Command Line

```
$ steps/align_si.sh --nj "$nj" --cmd "$train_cmd" \
   data/train data/lang exp/mono exp/mono_ali
$ steps/train_deltas.sh --boost-silence 1.25 --cmd "$train_cmd" \
   2000 20000 data/train data/lang exp/mono_ali exp/tri1
```

Question 2

`steps/train_deltas.sh` takes two arguments 2000 and 20000. Figure out for yourselves what these hyperparameters control. Tune them and observe how this tuning affects performance on the test set. Submit the final, tuned hyperparameter values within `run.sh` in the call to `steps/train_deltas.sh`. (If you have edited `steps/train_deltas.sh`, please submit `task2/train_deltas.sh` that we will use to train tied-state triphone HMMs.) Also submit a text file `task2/wer.txt` that only contains the best WER you obtained on `data/test` using your final tuned hyperparameters that we will aim to reproduce.

Task 3: Importance of speaker information [4 points]

Question 3

Within `data/train/utt2spk`, you will find each utterance mapped to a speaker ID. Edit this file such that each utterance has its own unique speaker ID. (You will also need to change the file `data/train/spk2utt` accordingly.) Rebuild tied-state triphone models as in task 2 using this new utterance-speaker mapping. Submit a text file `task3/wer.txt` with the resulting WER on `data/test`. (We will try to reproduce this at our end using your script `task2/train_deltas.sh` and your call to it from `run.sh`, with the speaker-to-utterance mapping changed.) Why did performance drop? Specifically, point to the steps within `task2/train_deltas.sh` that made use of speaker information and briefly explain how. Submit your answer as a text file, `task3/answer.txt`.

Task 4: Language models [4 points]

Now, revert back to the original `data/train/utt2spk` and `data/train/spk2utt`. For the baseline model, you were given a smoothed trigram LM in `corpus/LM/swahili.small.arpa`.

Question 4

Build different smoothed Ngram models (of higher order, with interpolation, etc.) with the help of SRILM tools trained on `corpus/LM/train.txt`. (More information about SRILM will be shared via Moodle.) Tune on the test set to identify the best of these language models via hyperparameter tuning and submit a file `task4/G.fst` that we can decode along with the best tied-state triphone models from task 2. Also, submit a text file `task4/wer.txt` with your best WER on `data/test` that we will aim to reproduce.
Hint: You will need to use `arpa2fst` to construct G.fst. You will find a call to `arpa2fst` within `local/prepare_lm.sh`.

Task 5: Augmentation [6 points]

Command Line

```
$ utils/data/perturb_data_dir_speed_3way.sh data/train data/train_sp3
```

Question 5

Explore the effect of data augmentation. This can be implemented with the help of speed perturbations in `utils/data/perturb_data_dir_speed_3way.sh` (mentioned in the command above). Include a new stage in `run.sh` within an if clause `if [ $stage -le 5 ]` to implement this data augmentation step. Reestimate tied-state triphone models using the augmented data and decode the newly estimated models within this new stage. Submit a text file `task5/wer.txt` with your best WER on `data/test` that we will aim to reproduce by running the new stage 5 in `run.sh`.

Task 6: Spot the phrase [6 points]

Question 6

For a given phrase, find all audio instances of the phrase in the training data. Submit a new script `task6/findphrase.sh` which, when run as `./findphrase.sh "lebuh maya kempen"`, will output a directory named `outputaudio` containing audio files `word01.wav`, `word02.wav`, etc., which are snippets of audio corresponding to the phrase `"lebuh maya kempen"`. You can use the commandline tool sox to snip an audio file. Any additional files or scripts you require should be within `task6`. (You can assume that the given phrase will be found at least once in the training data.)
(Hint: You will need to force-align the training utterances to its transcripts in order to find timestamps where the utterance can be snipped.)

Task 7: Speaker-specific WERs [4 points]

Question 7

Instead of providing a single WER for all the utterances in `data/test`, submit a script `task7/score.sh` (similar to `local/score.sh`) that produces WERs for utterances grouped by their speaker ID. Use your models from task 2 for evaluation. We will test your script on all the utterances in `data/test`.

Here is what a sample output should look like from running your script:

Command Line

```
ibf_002 22.30 [ 222 / 995, 100 ins, 12 del, 110 sub ]
ibm_006 38.40 [ 48/ 125, 22 ins, 5 del, 21 sub ]
```

Any other new script that your `score.sh` calls should be within `task7`.

Task 8: Performance on blind test set [4 points]

Question 8

This is the true test: How well does your Swahili ASR system perform on unseen utterances?

Check `corpus/data/truetest/wav.scp` for a sample set of unseen utterances. (We will replace this with a completely new set of unseen utterances during the blind test.) Create a new stage in `run.sh` within an if clause `if [ $stage -le 6 ]`, with your best trained models and decoding on `corpus/data/truetest`. Any files you need to run this new stage should be within `task8`. Any innovations are allowed in this new stage; the only requirement is that you should stick to estimating HMM-based acoustic models. A leaderboard with the top-scoring N roll numbers and the corresponding WERs will be posted on Moodle. (N will depend on where there's a clean split in WERs.) You will receive full points for this question if your WER on the unseen utterances is lower than what we get using the baseline recipe. The N top-scoring performers on the leaderboard will gain extra credit points.

Useful Kaldi resources

- How to install Kaldi using docker
- Kaldi tutorial for beginners.
- Kaldi lecture slides.
- Kaldi troubleshooting.