Kaldi Programming Assignment Part 2

DUE on 10/18/2016

In this part, you will be building a recognizer that recognizes digit words in English.

Data Description

We are going to use two corpora. The first one is a dataset called *TIDIGITS*. The data contains recordings from English speakers speaking up-to seven digits at a time. There are total 326 speakers, each says 77 (or sometimes a bit less) utterances (a sequence of digits), which sum up to 25096 utterances. (see the table below for detailed break-up) You can locate the corpus in the department workstation, placed at `/home/g/grad/krim/Corpus/LDC93S10-TIDIGITS/tidigits_comp` . The TIDIGITS dataset is around 800MB total. You can symlink it to your project directory if you work remotely, or you can download it to your personal workstation, but PLEASE DO NOT re-distribute it, as the data is licensed for research and educational use within Brandeis. Read the `readme.1st` file inside carefully before you start anything.

	man	woman	all adults	boy	girl	all children
test	4311	4389	8700	1925	1922	3847
train	4235	4388	8623	1925	2001	3926
total	8546	8777	17323	3850	3923	7773

The second one is rather smaller *FSDD*. This data only contains one speaker, speaking one digit at a time, but 50 times per a digit. You can download the data from the public github repository.

Preparing "*data*" portion

If you have followed the previous assignment, you won't have any trouble pre-processing this larger corpus for Kaldi training-decoding. The `readme.1st` file contains all the information you need to get transcripts. However you might want to carefully pick `utt_id` s as you need to repetitively train the acoustic model using different portions of the corpus. Also note that all recordings are in NIST sphere format, even though file extensions are `.wav` . So, in order to use the Kaldi feature extraction, you need to either convert audio files to MS wave format beforehand, or use `KALDI_ROOT/tools/sph2pipe_v2.5/sph2pipe` command to '*pipe*' the file into Kaldi front-end interface.

Preparing "*lang*" portion

There are no outside noise or out-of-vocab word appears in the recordings. So use "*silence*" as we did in the previous assignment.

Also we won't be using phonetic context or stresses/tones yet. That means we treat each phoneme in this small digits language has only a single realization, regardless of preceding/following sounds.

The tricky part is the lexicon. In this corpus, we have 11 non-silent words - `one`, `two`, `three`, `four`, `five`, `six`, `seven`, `eight`, `nine`, `zero`, and `oh` (more than five times larger than the one in the previous assignment!). Again, you can use any external resource to get their pronunciations, but we highly recommend you use CMU Pronouncing Dictionary. If you use Python to script your pipeline, NLTK has CMUdict bundled, so that would be helpful. (Ignore any variation for now, pick one if a word has multiple ways to pronounce. Also don't forget to take out stress markers if you're using CMUdict.)

Preparing language model

This time, there's no text data large enough to train a statistical language model, nor a pre-trained model. You need to hand-write your grammar using FST (using openfst format). Design a simple straight-forward finite state(s) and transition(s), don't worry about constraining the length of a sequence at 7. See openfst tutorial (part 2) and this section from the Kaldi documentation to figure how to write a grammar in a FST way, and how to compile it in a Kaldi way (to finally get `G.fst`). You will notice you need word indices to compile your grammar. They are in `OUTPUT/OF/PREPARE_LANG_SCRIPT/words.txt`.

Training and Decoding

Again, you don't have to use a fancy training or adaptation algorithm. Monophone training would work just fine as we saw in the previous assignment. However you will doing total 3 rounds of training-decoding, using different portions of the data.

1. train on train/woman + train/man, test on test/women + test/man
2. train on train/woman + train/man, test on FSDD
3. train on entire adults, test on test/girl + test/boy

For scoring you will be given `score.sh` script to get WER's, just like in the previous part. Note that this script will be automatically run by `decode.sh`.

Starter-kit and Project directory

We provide a starter-kit. Please feel free to use (or not to use) it as you want. However, the kit is structured in a Kaldi-convention way, and we want you to follow it as you work on the assignment.

```
ROOT
├──utils           # kaldi bundle scripts
├──steps           # kaldi bundle scripts
├──conf            # contains confurations for kaldi bundle scripts
├──local           # directory for your code
├──(data)          # kaldi bundle scripts
|   ├──(train_id)  # "data annotations" for a trainset
|   ├──(test_id)   # "data annotations" for a testset
|   ├──(lang)      # "language definitions" for the language
|   ├──(lang_id)   # use to store language models
|   |              # only when experimenting with multiple models
|   |              # otherwise, oaky to merge this with 'data/lang'
|   └──(tmp)       # any temporary stuff from the pipeline operations
├──(raw_data)        # use to store the raw data
|   ├──(tidigits)
|   └──(fsdd_waves)
├──(exp)           # to store output from kaldi operations
├──path.sh
└──run.sh          # an uber script to run-them-all
```

Given that you will perform multiple rounds of experiments with different subset of data, it is highly recommended to write the uber script, `run.sh`, to accept data locations (and any others necesssary) as its parameters.

Submission

Zip up your `run.sh`, `local` directory, and `exp/mono/decode*` directories for each round, and submit to the Latte submission box.

And as always, don't wait until the last minute, as the each training-decoding can take hours depending on your system performance.