# Kaldi lab using TIDIGITS

Michael Mandel, Vijay Peddinti, Shinji Watanabe
Based on a lab by Eric Fosler-Lussier

June 29, 2015

For this lab, we'll be following the Kaldi tutorial for building TIDIGITS. The lab will utilize a virtual machine for the VirtualBox host that contains all of the necessary software and data. All paths given in the lab are in the virtual machine's filesystem and all commands should be run in the virtual machine. The virtual machine requires hardware virtualization support, which is present in recent Intel and AMD CPUs, but might not be on older machines. If your machine does not support this, please join up with someone with a laptop that does.

Kaldi is a WFST-based speech recognizer – it builds four different WFST/WFSAs:

**H:** maps multiple HMM states (a.k.a. transition-ids in Kaldi-speak) to context-dependent triphones

**C:** maps triphone sequences to monophones

**L:** maps monophone sequences to words

**G:** FSA grammar (can be built from an n-gram grammar).

For more detail, see the Kaldi documentation page (`http://kaldi.sourceforge.net`) and, particularly:

**Tutorial** `http://kaldi.sourceforge.net/tutorial.html`

**Data input files** `http://kaldi.sourceforge.net/data_prep.html`

**Decoding graph** `http://kaldi.sourceforge.net/graph.html`

Your job today is to build a digit recognizer, following the script that comes with Kaldi.

Note: the audio data is stored in `~/data/tidigits`. This data is stored in NIST wave file format (which is different than the Microsoft WAV format). The filename of each file gives the transcription, of the form `##...#a.wav` or `##...#b.wav`. So `z12a.wav` contains the phrase "zero one two" and `34oa.wav` contains "three four oh". The suffixes `a` and `b` distinguish multiple repetitions of the same digit string from one another.

# 1 VirtualBox setup (please attempt this ahead of time)

## 1.1 Download and install VirtualBox host software

VirtualBox is available for all major platforms from https://www.virtualbox.org/wiki/Downloads.

On Ubuntu, you can install it using apt-get:

```
sudo apt-get install virtualbox
```

## 1.2 Linux: Install exfat tools

The USB key uses the exfat filesystem. Exfat support is built-in to Windows and Mac OSX 10.6.5 and higher, but on Linux you might need to install drivers for it. On Ubuntu, you can call

```
sudo apt-get install exfat-utils exfat-fuse
```

## 1.3 Download Prix Fixe virtual machine image

Download the virtual machine image for the "Prix Fixe" machine from the speech recognition virtual kitchen project at http://speechkitchen.org/prix-fixe-vm/ . The image is 1.7 GB, so it might take a while to download and take up a lot of your disk space once it is installed.

## 1.4 Import virtual image

Start VirtualBox and import the image into your VirtualBox library by going to the File → Import Appliance menu and selecting the file `Ilva-Prix-FixeV1.0.ova`. It will take a few minutes to import the image.

## 1.5 Attempt to boot the virtual machine

In the VirtualBox Manager, select the `SRVK_Education3` image and click the "Start" button. If the machine boots up, great. If it instead gives you an error message, follow the steps below to resolve it.

### 1.6 Resolve error with ethernet adapter

If you get an error message about a missing ethernet adapter, choose the default resolution. Attempt to boot the machine again.

### 1.7 Resolve error with USB driver

If you get an error message about an error with the USB driver, install the VirtualBox extension pack from the Oracle website: https://www.virtualbox.org/wiki/Downloads. Attempt to boot the machine again.

### 1.8 Resolve error about hardware virtualization not being enabled in your BIOS

If you get an error message that "VT-x is disabled in the BIOS" not being enabled in your BIOS, you will need to reboot your laptop into the BIOS setup (usually by holding down F2 or F12 at boot). Go to the appropriate menu and turn on hardware virtualization support. This may not be possible on older machines.

Reboot your computer, start up VirtualBox, and attempt to boot the virtual machine again.

### 1.9 Successfully start your virtual machine

Congratulations! At this point, your virtual machine should be up and running and you should see an Ubuntu desktop. If so, shut down the virtual machine and bring your configured laptop to the lab. If not, bring it to the lab and we can probably help you get it running in person.

## 2 Import the actual lab machine (in-lab)

### 2.1 Get a USB key from us

There are 5 USB keys with the virtual machine image that we are going to use for the lab. Please get one from us, and plug it into your computer.

### 2.2 Import virtual image

Start VirtualBox and import the image directly from the USB key into your VirtualBox library. Do this by going to the File → Import Appliance menu

and selecting the file `jsaltSrvk-ubuntu-v4.ova`. It will take a few minutes to import the image.

### 2.3 Ignore a warning about a shared folder not existing

You can fix it if you want by going into Settings → Shared Folders and selecting a directory that exists.

### 2.4 Successfully start your virtual machine

At this point, your virtual machine should be up and running and you should see an Ubuntu desktop. If not, let us know and we can help you debug it further.

### 2.5 Go to the right directory

Now start a terminal using the `gnome-terminal` command and go to the right directory:

```
cd ~/src/kaldi/kaldi-trunk/egs/tidigits/s5
```

## 3 Step through `run.sh`

Look in `run.sh` . This gives a complete set of commands to go through in building a complete recognizer. While you can run all of these commands just by running `./run.sh`, you may find it more instructive to cut and paste commands to see what is going on.

Below is a blow-by-blow of what is going on. Code blocks are in gray rectangles with corresponding line numbers from `./run.sh`. Code blocks without line numbers are commands you should run, but are not in `./run.sh`.

### 3.1 Set up some environment variables

The script `./cmd.sh` sets up the environment variables `$train_cmd` and `$decode_cmd` with command appropriate for your computing environment. There are different versions that you can use for different cluster setups. The current version runs everything locally.

```
7  . ./path.sh
8  . ./cmd.sh
```

### 3.2 Set a variable to track where the data is stored

```
15 tidigits=/home/mario/data/tidigits
```

### 3.3 Setup some files that are needed

Run this script, then read it to see what's going on:

```
18 local/tidigits_data_prep.sh $tidigits || exit 1;
```

This creates training and test lists, for example

- `data/train/spk2utt` lists all of the utterances for a speaker in the training set
- `data/train/text` gives the transcription for each utterance
- `data/train/utt2spk` lists the speaker for each utterance
- `data/train/wav.scp` a script that will create input (via a linux pipe) for each waveform
- `data/local/data/train.flist` list of training files
- `data/local/data/train_sph.scp` mapping of each utterance to the corresponding file

There are similar files for the test set.

### 3.4 Create the language model

```
19 local/tidigits_prepare_lang.sh  || exit 1;
```

Notice that the dictionary is hard-coded in the script, which is a bit unusual for this setup. This creates the FSTs for the lexicon and grammar. Read through the script and see what has been created. You should be able to use your OpenFST sleuthing skills.

### 3.5 Validate the language models

```
20 utils/validate_lang.pl data/lang/
```

As the script says, this will have some errors because the system expects there to be disambiguation symbols. In general, you need to have disambiguation symbols when you have one word that is a prefix of another (cat and cats in the same lexicon would need to have cat being pronounced "k ae t #1") or a homophone of another word (red: "r eh d #1", read: "r eh d #2"). If you don't have these then the models become nondeterministic.

### 3.6 Create the MFCCs from the wavefile lists

The option `--nj` refers to the number of parallel jobs; on a laptop you'll want to reduce this to 2 or 4, depending on the number of cores you have.

```
mfccdir=mfcc
for x in test train; do
 steps/make_mfcc.sh --cmd "$train_cmd" --nj 20 \
   data/$x exp/make_mfcc/$x $mfccdir || exit 1;
 steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir || exit 1;
done
```

This will take a while. Look through the MFCC creation script in the meantime. Note that it calls on a config file in `conf/mfcc.conf`. Also, let's talk about what CMVN does: this stands for Cepstral Mean and Variance Normalization. Many systems often find it useful to subtract the mean cepstral vector and divide each dimension by the variance – this means that the resulting vectors have mean $\mathbf{0}$ and covariance matrix $I$ (or, variance $\mathbf{1}$). This recipe performs CMVN per speaker, so the mean across all of the utterances of each speaker should be 0. Each speaker in the test set will be normalized by a different vector – this can help reduce the mismatches between training and test sets.

### 3.7 Fix error (outside of `run.sh`)

OK, now that you've read that, an error cropped up, and one file didn't generate any data. So fix it using:

```
utils/fix_data_dir.sh data/test
```

### 3.8 Create smaller bootstrap training set

The Kaldi system bootstraps by starting with a smaller training set, which you create by doing

```
utils/subset_data_dir.sh data/train 1000 data/train_1k
```

### 3.9 Build monophone model from flat start

The first thing the system does is to build a monophone model from a "flat start" – that is assume that the monophones are all equally likely and all have the same means and variances (those of the observations globally). The fact that we have transcripts to force-align to allows us to break that symmetry. It builds from the `train_1k` dataset. (Remember to adjust the number of jobs.)

```
steps/train_mono.sh --nj 2 --cmd "$train_cmd" \
    data/train_1k data/lang exp/mono0a
```

While this is running, open up the `train_mono.sh` script and take a look at it. The first critical step is `gmm-init-mono`, which initializes the monophones. The model file is binary, but you can convert it to text by doing the following:

```
/home/mario/src/kaldi/kaldi-trunk/src/bin/copy-transition-model \
    --binary=false exp/mono0a/0.mdl exp/mono0a/0.txt
```

See if you can piece together what is happening in here. It's a bit complex. The key idea is that multiple passes of EM training are done over the data. Compare this to the trained model (after it finishes):

```
/home/mario/src/kaldi/kaldi-trunk/src/bin/copy-transition-model \
    --binary=false exp/mono0a/final.mdl exp/mono0a/final.txt
```

You can also look at the alignments that are created:

```
../../../src/bin/show-alignments data/lang/phones.txt exp/mono0a/final.mdl \
    "ark:gunzip -c exp/mono0a/ali.2.gz|" | head -5 | less
```

### 3.10 Decode the test set using this initial model

```
utils/mkgraph.sh --mono data/lang exp/mono0a exp/mono0a/graph && \
    steps/decode.sh --nj 2 --cmd "$decode_cmd" \
    exp/mono0a/graph data/test exp/mono0a/decode
```

### 3.11 Align the entire training set using the "small" model

```
steps/align_si.sh --nj 4 --cmd "$train_cmd" \
    data/train data/lang exp/mono0a exp/mono0a_ali
```

### 3.12 Use this alignment to train triphones

```
steps/train_deltas.sh --cmd "$train_cmd" \
    300 3000 data/train data/lang exp/mono0a_ali exp/tri1
```

Again, it might be useful to look through the script to see what's going on. There are alternate passes of EM training and alignment.

### 3.13 Decode test set with trigram model

Finally, after training, the test set can be decoded using the trigram model:

```
utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph
steps/decode.sh --nj 2 --cmd "$decode_cmd" \
    exp/tri1/graph data/test exp/tri1/decode
```

What the decode does (for both the monophone and triphone cases) is to run the decoding several times with different language model weights. The acoustic model scores, since they have quite a few frames in them, tend to be on a different scale than the language model scores (which are over words). So we scale the probabilities in the language model by multiplying the log probability by a weight in order to bring it into line with the range of the acoustic model scores.

### 3.14 Examine results

Now check out the results – the system computes both a Word Error Rate (WER) and Sentence Error Rate (SER). The easiest way to find the best language model scale is to use a utility script:

```
# for x in exp/*/decode*; do [ -d $x ] && grep SER $x/wer_* | utils/best_wer.sh; done
```

which gives:

```
#exp/mono0a/decode/wer_17:%SER 3.67 [ 319 / 8700 ]
#exp/tri1/decode/wer_19:%SER 2.64 [ 230 / 8700 ]
```

So you can see that this builds a relatively accurate digit recognizer (only 2.64% of digit strings are wrong, although your number might be slightly different).

## 4 Extensions

If you've made it this far, congratulations! Spend the rest of the time in the lab extending the TIDIGITS recognizer, here are some suggestions, do as many as time permits. Remember to provide comparison results to the baseline.

- Change the number of states per phone in the HMM
- Build whole word models (e.g., the pronunciation of 'one' is "one" instead of "w ah n". You may need to increase the number of states.
- Pronunciations become specific to words – e.g., replace four "f ao r" with "41 42 43", and five "f ay v" with "51 52 53"
- Train on nested subsets of the training data, report results as a function of training data size
- Download the noisy digit set from http://www1.icsi.berkeley.edu/Speech/mr/mrdigits.html and build a system with that.

- Conduct an experiment of your choosing.