Kaldi for Dummies tutorial

Table of Contents

Introduction

This is a step by step tutorial for absolute beginners on how to create a simple ASR (Automatic Speech Recognition) system in Kaldi toolkit using your own set of data. I really would have liked to read something like this when I was starting to deal with Kaldi. This is all based on my experience as an amateur in case of speech recognition subject and script programming as well. If you have ever delved through Kaldi tutorial on the official project site and felt a little bit lost, well, my piece of art might be the choice for you. You will learn how to install Kaldi, how to make it work and how to run an ASR system using your own audio data. As an effect you will get your first speech decoding results. It was created by Wit Zielinski.

First of all - get to know what Kaldi actually is and why you should use it instead of something else. In my opinion Kaldi requires solid knowledge about speech recognition and ASR systems in general. It is also good to know the basics of script programming languages (bash, perl, python). C++ migh be useful in the future (probably you will want to make some modifications in the source code).

To read:
About the Kaldi project
Kaldi tutorial: Prerequisites

Environment

Rule number 1 - use Linux. Although it is possible to use Kaldi on Windows, most people I find trustworthy convinced me that Linux will do the job with the less amount of problems. I have chosen Ubuntu 14.10. This was (in 2014/15) a rich and stable Linux representation which I honestly recommend. When you finally have your Linux running properly, please open a terminal and install some necessary stuff (if you do not already have it):

(has to be installed)

- `atlas` – automation and optimization of calculations in the field of linear algebra,
- `autoconf` – automatic software compilation on different operating systems,
- `automake` – creating portable Makefile files,
- `git` – distributed revision control system,
- `libtool` – creating static and dynamic libraries,
- `svn` – revision control system (Subversion), necessary for Kaldi download and installation,
- `wget` – data transfer using HTTP, HTTPS and FTP protocols,
- `zlib` – data compression,

(probably has to be installed)

- `awk` – programming language, used for searching and processing patterns in files and data streams,
- `bash` – Unix shell and script programming language,
- `grep` – command-line utility for searching plain-text data sets for lines matching a regular expression,
- `make` – automatically builds executable programs and libraries from source code,
- `perl` – dynamic programming language, perfect for text files processing.

Done. Operating system and all the necessary Linux tools are ready to go.

Download Kaldi

Just follow the instruction carefully: **Downloading and installing Kaldi**. If you do not have much idea about how to use GIT, please read about it: **Kaldi Tutorial: Version control with Git (5 minutes)**.

I installed Kaldi in this directory (called 'Kaldi root path'): `/home/{user}/kaldi-trunk`

Kaldi directories structure

Try to acknowledge where particular Kaldi components are placed. Also it would be nice if you read any `README` files you find.

`kaldi-trunk` - main Kaldi directory which contains:

- `egs` – example scripts allowing you to quickly build ASR systems for over 30 popular speech corporas (documentation is attached for each project),
- `misc` – additional tools and supplies, not needed for proper Kaldi functionality,
- `src` – Kaldi source code,
- `tools` – useful components and external tools,
- `windows` – tools for running Kaldi using Windows.

The most important directory for you is obviously `egs`. Here you will create your own ASR system.

Your exemplary project

For the purpose of this tutorial, imagine that you have the same simple set of data as me (described below, in **Audio data** section). Then try to 'transpose' every action I do straight into your own project. If you completely do not have any audio data or you want to follow my tutorial in an identical way, feel free to record your own tracks - it will be even bigger experience to play with ASR. Here we go.

Your precondition

You have some amount of audio data that contain only spoken digits by at least several different speakers. Each audio file is an entire spoken sentence (e.g. 'one, nine, five').

Your purpose

You want to divide your data into train and test sets, set up an ASR system, train it, test it and get some decoding results.

Your first task

Something to begin with - create a folder `digits` in `kaldi-trunk/egs/` directory. This is a place where you will put all the stuff related to your project.

Data preparation

Audio data

I assume that you want to set up an ASR system, basing on your own audio data. For example - let it be a set of 100 files. File format is WAV. Each file contains 3 spoken digits recorded in english language, one by one. Each of these audio files is named in a recognizable way (e.g. `1_5_6.wav`, which in my pattern means that the spoken sentence is 'one, five, six') and placed in the recognizable folder representing particular speaker during a particular recording session (there may be a situation that you have recordings of the same person but in two different quality/noise environments - put these in separate folders). So to sum up, my exemplary data set looks like this:

- 10 different speakers (ASR systems must be trained and tested on different speakers, the more speakers you have the better),
- each speaker says 10 sentences,
- 100 senteces/utterances (in 100 *.wav files placed in 10 folders related to particular speakers - 10 *.wav files in each folder),
- 300 words (digits from zero to nine),
- each sentence/utterance consist of 3 words.

Whatever your first data set is, adjust my example to your particular case. Be careful with big data sets and complex grammars - start with something simple. Sentences that contain only digits are perfect in this case.

Task

Go to `kaldi-trunk/egs/digits` directory and create `digits_audio` folder. In `kaldi-trunk/egs/digits/digits_audio` create two folders: `train` and `test`. Select one speaker of your choice to represent testing data set. Use this speaker's 'speakerID' as a name for an another new folder in `kaldi-trunk/egs/digits/digits_audio/test` directory. Then put there all the audio files related to that person. Put the rest (9 speakers) into `train` folder - this will be your training data set. Also create subfolders for each speaker.

Acoustic data

Now you have to create some text files that will allow Kaldi to communicate with your audio data. Consider these files as 'must be done'. Each file that you will create in this section (and in **Language data** section as well) can be considered as a text file with some number of strings (each string in a new line). These strings need to be sorted. If you will encounter any sorting issues you can use Kaldi scripts for checking (`utils/validate_data_dir.sh`) and fixing (`utils/fix_data_dir.sh`) data order. And for you information - `utils` directory will be attached to your project in **Tools attachment** section.

Task

In `kaldi-trunk/egs/digits` directory, create a folder `data`. Then create `test` and `train` subfolders inside. Create in each subfolder following files (so you have files named in **the same way in `test` and**

`train` subfolders but they relate to two different data sets that you created before):

a.) `spk2gender`
This file informs about speakers gender. As we assumed, 'speakerID' is a unique name of each speaker (in this case it is also a 'recordingID' - every speaker has only one audio data folder from one recording session). In my example there are 5 female and 5 male speakers (f = female, m = male).

Pattern: <speakerID> <gender>

```
cristine f
dad m
josh m
july f
# and so on...
```

b.) `wav.scp`
This file connects every utterance (sentence said by one person during particular recording session) with an audio file related to this utterance. If you stick to my naming approach, 'utteranceID' is nothing more than 'speakerID' (speaker's folder name) glued with *.wav file name without '.wav' ending (look for examples below).

Pattern: <uterranceID> <full_path_to_audio_file>

```
dad_4_4_2 /home/{user}/kaldi-trunk/egs/digits/digits_audio/train/dad/4_4_2.wa
july_1_2_5 /home/{user}/kaldi-trunk/egs/digits/digits_audio/train/july/1_2_5.
july_6_8_3 /home/{user}/kaldi-trunk/egs/digits/digits_audio/train/july/6_8_3.
# and so on...
```

c.) `text`
This file contains every utterance matched with its text transcription.

Pattern: <uterranceID> <text_transcription>

```
dad_4_4_2 four four two
july_1_2_5 one two five
july_6_8_3 six eight three
# and so on...
```

d.) `utt2spk`
This file tells the ASR system which utterance belongs to particular speaker.

Pattern: <uterranceID> <speakerID>

```
dad_4_4_2 dad
july_1_2_5 july
july_6_8_3 july
# and so on...
```

e.) `corpus.txt`
This file has a slightly different directory. In `kaldi-trunk/egs/digits/data` create another folder `local`. In `kaldi-trunk/egs/digits/data/local` create a file `corpus.txt` which should contain every

single utterance transcription that can occur in your ASR system (in our case it will be 100 lines from 100 audio files).

Pattern: <text_transcription>

```
one two five
six eight three
four four two
# and so on...
```

Language data

This section relates to language modelling files that also need to be considered as 'must be done'. Look for the syntax details here: **Data preparation** (each file is precisely described). Also feel free to read some examples in other egs scripts. Now is the perfect time.

Task

In `kaldi-trunk/egs/digits/data/local` directory, create a folder `dict`. In `kaldi-trunk/egs/digits/data/local/dict` create following files:

a.) `lexicon.txt`
This file contains every word from your dictionary with its 'phone transcriptions' (taken from `/egs/voxforge`).

Pattern: <word> <phone 1> <phone 2> ...

```
!SIL sil
<UNK> spn
eight ey t
five f ay v
four f ao r
nine n ay n
one hh w ah n
one w ah n
seven s eh v ah n
six s ih k s
three th r iy
two t uw
zero z ih r ow
zero z iy r ow
```

b.) `nonsilence_phones.txt`
This file lists nonsilence phones that are present in your project.

Pattern: <phone>

```
ah
ao
ay
eh
ey
f
hh
```

```
ih
iy
k
n
ow
r
s
t
th
uw
w
v
z
```

c.) `silence_phones.txt`
This file lists silence phones.

Pattern: <phone>

```
sil
spn
```

d.) `optional_silence.txt`
This file lists optional silence phones.

Pattern: <phone>

```
sil
```

Project finalization

Last chapter before runnig scripts creation. Your project structure will become complete.

Tools attachment

You need to add necessary Kaldi tools that are widely used in exemplary scripts.

Task

From `kaldi-trunk/egs/wsj/s5` copy two folders (with the whole content) - `utils` and `steps` - and put them in your `kaldi-trunk/egs/digits` directory. You can also create links to these directories. You may find such links in, for example, `kaldi-trunk/egs/voxforge/s5`.

Scoring script

This script will help you to get decoding results.

Task

From `kaldi-trunk/egs/voxforge/s5/local` copy the script `score.sh` into similar location in your project (`kaldi-trunk/egs/digits/local`).

SRILM installation

You also need to install language modelling toolkit that is used in my example - SRI Language Modeling Toolkit (SRILM).

Task

For detailed installation instructions go to `kaldi-trunk/tools/install_srilm.sh` (read all comments inside).

Configuration files

It is not necessary to create configuration files but it can be a good habit for future.

Task

In `kaldi-trunk/egs/digits` create a folder `conf`. Inside `kaldi-trunk/egs/digits/conf` create two files (for some configuration modifications in decoding and mfcc feature extraction processes - taken from `/egs/voxforge`):

a.) `decode.config`

```
first_beam=10.0
beam=13.0
lattice_beam=6.0
```

b.) `mfcc.conf`

```
--use-energy=false
```

Running scripts creation

Your first ASR system written in Kaldi environment is almost ready. Your last job is to prepare running scripts to create ASR system of your choice. I put some comments in prepared scripts for ease of understanding.

These scripts are based on solution used in `/egs/voxforge` directory. I decided to use two different training methods:

- MONO - monophone training,
- TRI1 - simple triphone training (first triphone pass).

These two methods are enough to show noticable differences in decoding results using only digits lexicon and small training data set.

Task

In `kaldi-trunk/egs/digits` directory create 3 scripts:

a.) `cmd.sh`

```
# Setting local system jobs (local CPU - no external clusters)
export train_cmd=run.pl
export decode_cmd=run.pl
```

b.) `path.sh`

```
# Defining Kaldi root directory
export KALDI_ROOT=`pwd`/../..

# Setting paths to useful tools
export PATH=$PWD/utils/:$KALDI_ROOT/src/bin:$KALDI_ROOT/tools/openfst/bin:$KALDI_ROOT/src/fstbin/:$KALDI_ROOT/src/gmmbin/:$KALDI_ROOT/src/featbin/:$KALDI_ROOT/src/lmbin/:$KALDI_ROOT/src/sgmm2bin/:$KALDI_ROOT/src/fgmmbin/:$KALDI_ROOT/src/latbin/:$PWD:$PATH

# Defining audio data directory (modify it for your installation directory!)
export DATA_ROOT="/home/{user}/kaldi-trunk/egs/digits/digits_audio"

# Enable SRILM
source $KALDI_ROOT/tools/env.sh

# Variable needed for proper data sorting
export LC_ALL=C
```

c.) `run.sh`

```
#!/bin/bash

. ./path.sh || exit 1
. ./cmd.sh || exit 1

nj=1       # number of parallel jobs - 1 is perfect for such a small data set
lm_order=1 # language model order (n-gram quantity) - 1 is enough for digits grammar

# Safety mechanism (possible running this script with modified arguments)
. utils/parse_options.sh || exit 1
[[ $# -ge 1 ]] && { echo "Wrong arguments!"; exit 1; }

# Removing previously created data (from last run.sh execution)
rm -rf exp mfcc data/train/spk2utt data/train/cmvn.scp data/train/feats.scp data/train/split1 data/test/spk2utt data/test/cmvn.scp data/test/feats.scp data/test/split1 data/local/lang data/lang data/local/tmp data/local/dict/lexiconp.txt

echo
echo "===== PREPARING ACOUSTIC DATA ====="
echo

# Needs to be prepared by hand (or using self written scripts):
#
# spk2gender  [<speaker-id> <gender>]
# wav.scp     [<uterranceID> <full_path_to_audio_file>]
# text           [<uterranceID> <text_transcription>]
# utt2spk     [<uterranceID> <speakerID>]
# corpus.txt  [<text_transcription>]

# Making spk2utt files
utils/utt2spk_to_spk2utt.pl data/train/utt2spk > data/train/spk2utt
utils/utt2spk_to_spk2utt.pl data/test/utt2spk > data/test/spk2utt

echo
echo "===== FEATURES EXTRACTION ====="
echo

# Making feats.scp files
mfccdir=mfcc
# Uncomment and modify arguments in scripts below if you have any problems with data sorting
# utils/validate_data_dir.sh data/train     # script for checking prepared data - here: for data/train directory
# utils/fix_data_dir.sh data/train          # tool for data proper sorting if needed - here: for data/train directory
steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/train exp/make_mfcc/train $mfccdir
steps/make_mfcc.sh --nj $nj --cmd "$train_cmd" data/test exp/make_mfcc/test $mfccdir

# Making cmvn.scp files
steps/compute_cmvn_stats.sh data/train exp/make_mfcc/train $mfccdir
```

```
steps/compute_cmvn_stats.sh data/test exp/make_mfcc/test $mfccdir

echo
echo "===== PREPARING LANGUAGE DATA ====="
echo

# Needs to be prepared by hand (or using self written scripts):
#
# lexicon.txt           [<word> <phone 1> <phone 2> ...]
# nonsilence_phones.txt    [<phone>]
# silence_phones.txt    [<phone>]
# optional_silence.txt  [<phone>]

# Preparing language data
utils/prepare_lang.sh data/local/dict "<UNK>" data/local/lang data/lang

echo
echo "===== LANGUAGE MODEL CREATION ====="
echo "===== MAKING lm.arpa ====="
echo

loc=`which ngram-count`;
if [ -z $loc ]; then
   if uname -a | grep 64 >/dev/null; then
           sdir=$KALDI_ROOT/tools/srilm/bin/i686-m64
   else
                   sdir=$KALDI_ROOT/tools/srilm/bin/i686
   fi
   if [ -f $sdir/ngram-count ]; then
                   echo "Using SRILM language modelling tool from $sdir"
                   export PATH=$PATH:$sdir
   else
                   echo "SRILM toolkit is probably not installed.
                           Instructions: tools/install_srilm.sh"
                   exit 1
   fi
fi

local=data/local
mkdir $local/tmp
ngram-count -order $lm_order -write-vocab $local/tmp/vocab-full.txt -wbdiscount -text $local/corpus.txt -lm $local/tmp/lm.arpa

echo
echo "===== MAKING G.fst ====="
echo

lang=data/lang
arpa2fst --disambig-symbol=#0 --read-symbol-table=$lang/words.txt $local/tmp/lm.arpa $lang/G.fst

echo
echo "===== MONO TRAINING ====="
echo

steps/train_mono.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono || exit 1

echo
echo "===== MONO DECODING ====="
echo

utils/mkgraph.sh --mono data/lang exp/mono exp/mono/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" exp/mono/graph data/test exp/mono/decode

echo
echo "===== MONO ALIGNMENT ====="
echo

steps/align_si.sh --nj $nj --cmd "$train_cmd" data/train data/lang exp/mono exp/mono_ali || exit 1

echo
echo "===== TRI1 (first triphone pass) TRAINING ====="
echo

steps/train_deltas.sh --cmd "$train_cmd" 2000 11000 data/train data/lang exp/mono_ali exp/tri1 || exit 1

```

```
echo
echo "===== TRI1 (first triphone pass) DECODING ====="
echo

utils/mkgraph.sh data/lang exp/tri1 exp/tri1/graph || exit 1
steps/decode.sh --config conf/decode.config --nj $nj --cmd "$decode_cmd" exp/tri1/graph data/test exp/tri1/decode

echo
echo "===== run.sh script is finished ====="
echo
```

Getting results

Now all you have to do is to run `run.sh` script. If I have made any mistakes in this tutorial, logs from the terminal should guide you how to deal with it.

Besides the fact that you will notice some decoding results in the terminal window, go to newly made `kaldi-trunk/egs/digits/exp`. You may notice there folders with `mono` and `tri1` results as well - directories structure are the same. Go to `mono/decode` directory. Here you may find result files (named in a `wer_{number}` way). Logs for decoding process may be found in `log` folder (same directory).

Summary

This is just an example. The point of this short tutorial is to show you how to create 'anything' in Kaldi and to get a better understanding of how to think while using this toolkit. Personally I started with looking for tutorials made by the Kaldi authors/developers. After succesful Kaldi installation I launched some example scripts (Yesno, Voxforge, LibriSpeech - they are relatively easy and have free acoustic/language data to download - I used these three as a base for my own scripts).

Make sure you follow http://kaldi-asr.org/- official project website. There are two very useful sections for beginners inside:
a.) **Kaldi tutorial** - almost 'step by step' tutorial on how to set up an ASR system; up to some point this can be done without RM dataset. It is good to read it,
b.) **Data preparation** - very detailed explaination of how to use your own data in Kaldi.

More useful links about Kaldi I found:
https://sites.google.com/site/dpovey/kaldi-lectures - Kaldi lectures created by the main author
http://www.superlectures.com/icassp2011/category.php?lang=en&id=131 - similar; video version
http://www.diplomovaprace.cz/133/thesis_oplatek.pdf - some master diploma thesis about speech recognition using Kaldi

This is all from my side. Good luck!