L23: Kaldi

Introduction to Kaldi

Structure of Kaldi directories

Building a demo ASR engine in Kaldi

This lecture is based on lecture notes by Karel Vesely, a tutorial by Eleanor Chodroff, and the official tutorial "Kaldi for Dummies"

Introduction

What is Kaldi?

- A toolkit for ASR written in C++ with Apache License v2.0
- Intended for use by speech recognition researchers
- Largely maintained by Dan Povey (JHU), the main architect

Kaldi vs. HTK

- Kaldi is relatively new (2011) compared to HTK (1996)
- Kaldi is based on WFSTs, whereas HTK focuses on HMMs
- Kaldi has a huge developer and user community
- Kaldi includes recipes for multiple ASR tasks (a huge advantage)
- Kaldi's Apache v2.0 license allows modification and commercial use; HTK cannot be used in commercial software

History

- 2009: initial software development at a JHU workshop
- 2010-2013: summer workshops are held to further develop Kaldi
- 2011: Kaldi code is released
- 2015: Kaldi moves to GitHub

The Kaldi GitHub[1] project contains

- Command-line programs to build ASR models
- Example recipes for single or cluster computers
- CUDA matrix libraries
- Extensive documentation
- Support forum

Kaldi recipes[2]

- Its main advantage, compared to other toolkits (HTK, Sphinx…)
- Toy examples (yes/no, TIDIGITS) and free databases
- Standard tasks on easy/difficult tasks (requires paid data)
  - Read speech (Resource Management, TIMIT, WSJ), WER: 2-4%
  - Conversational telephone speech (Switchboard), WER: 10%
  - Spontaneous ‘microphone array’ speech (AMI meetings), WER: 20-30%

[1] https://github.com/kaldi-asr/kaldi

[2] http://kaldi-asr.org/doc/examples.html

Speech processing techniques in Kaldi

- Feature extraction: MFCC, PLP
- Acoustic models: GMMs, SGMMs, DNNs
- Language models: n-grams, RNN-LM
- Speaker adaptation: CMVM, VTLN, fMLLR, i-vector
- HMM decoder using WFSTs (based on OpenFST library)
- A huge list of tools[3]: clustering, trees, alignment, decoding, transforms (LDA, PCA, affine…)

[3] http://kaldi-asr.org/doc/tools.html

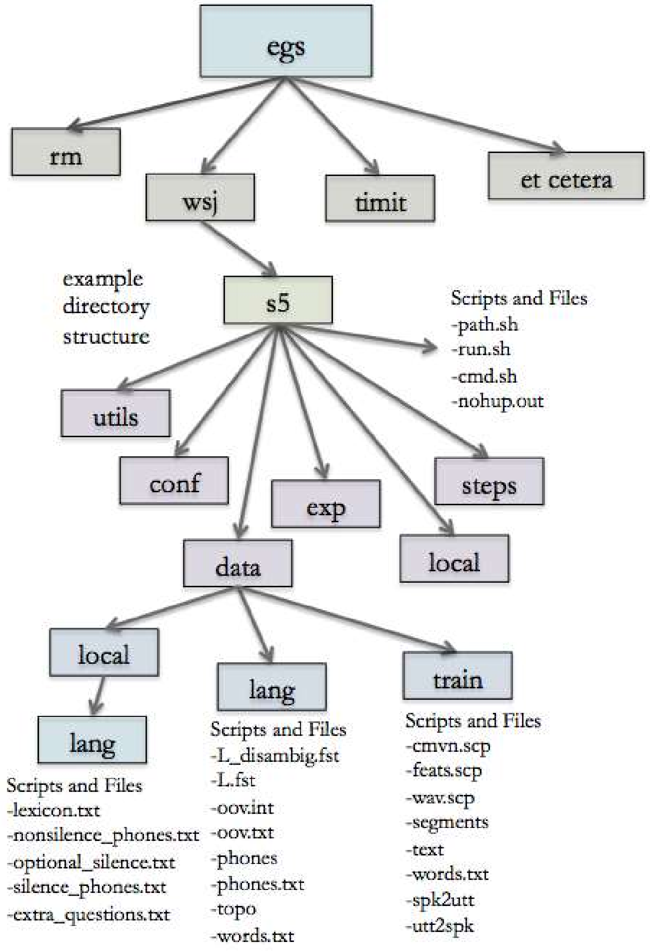
Kaldi’s directory structure

Top level directories

- `egs`: training recipes for multiple corpora
  - The most recent version of the recipes is under directory ‘s5’
- `src`: Kaldi source code
- `tools`: useful components, external tools,
- `misc`: additional tools and supplies, not needed for proper Kaldi functionality,
- `windows`: running Kaldi using Windows

Within each ‘egs’ directory

- Scripts
  - **`run.sh`**: the main training recipe
  - `cmd.sh` and `path.sh` (may need editing)
- Directories
  - `conf`: configuration files
  - `data`, `exp`: for experiments
  - `local`, `steps`, `utils`: links to utilities



https://www.eleanorchodroff.com/tutorial/kaldi/kaldi-familiarization.html

Training acoustic models

A typical process for building acoustic models consists of:

- Record speech corpora
- Format transcripts for Kaldi
- Extract acoustic features from recordings
- Train mono-phone models
- Align audio with the acoustic models
- Train tri-phone models
- Re-align audio with the acoustic models
- Re-train tri-phone models

Building a demo ASR (aka "Kaldi for dummies")

The task

- Recognition of three digit numbers, e.g., "one, two, eight"
  - 10 different speakers, 10 sentences per speaker
  - Thus, 100 sentences, each on a separate wav file
  - Each sentence consists of three words

Create the speech corpus

- Collect the recordings
  - You may use PRAAT, audacity or any other tool
  - Name each file with a descriptive name, e.g., `1_2_8.wav`
  - Put each speaker's recording on a unique folder, e.g., `christine`
- Create the following folders[1]
  - `kaldi-trunk/egs/digits`
  - `kaldi-trunk/egs/digits/digits_audio`
  - `kaldi-trunk/egs/digits/digits_audio/train`
  - `kaldi-trunk/egs/digits/digits_audio/test`
- Move N speakers to `test`, and the rest to `train`

[1] Replace '`kaldi-trunk`' with the exact path to your Kaldi installation

Prepare files for the acoustic data

- Create the following folders
  - `kaldi-trunk/egs/digit/data`
  - `kaldi-trunk/egs/digit/data/train`
  - `kaldi-trunk/egs/digit/data/test`

In all these files, make sure that there are
- no extra spaces at the end of each line, and
- not extra lines at the end of the file

- Under `train`, create the following files

spk2gender

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

text

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

utt2spk

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

wav.scp

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

  - Repeat the process for all the folders under `test`

- Finally:
  - Create folder `kaldi-trunk/egs/digits/data/local`
  - And inside it create the file `corpus.txt`

corpus.txt

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

Prepare language data

- Create folder `dict` under `kaldi-trunk/egs/digits/data/local`
- Inside it, create the following files

lexicon.txt

```
!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
```

nonsilence_phones.txt

```
ah
ao
ay
eh
ey
f
hh
ih
iy
k
n
ow
r
s
t
th
uw
w
v
z
```

silence_phones.txt

```
sil
spn
```

optional_silence.txt

```
sil
```

Setting up the tools and scripts

- In `kaldi-trunk/egs/digits`, create links to folders `utils` and `steps` in `kaldi-trunk/egs/wsj/s5`
- From `kaldi-trunk/egs/voxforge/s5/local` copy the script `score.sh` into similar location in your project
- Install the SRI language modeling toolkit
  - For detailed installation instructions go to `kaldi-trunk/tools/install_srilm.sh` (read all comments inside)
- Configuration files
  - Create folder `conf` inside `kaldi-trunk/egs/digit`
  - Inside of it, create two files:

decode.config

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

mfcc.conf

```
--use-energy=false
--sample_frequency=44100
```

Modify to match the sampling rate in your recordings

Preparing the training scripts

– In `kaldi-trunk/egs/digits` create three scripts

cmd.sh

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

path.sh

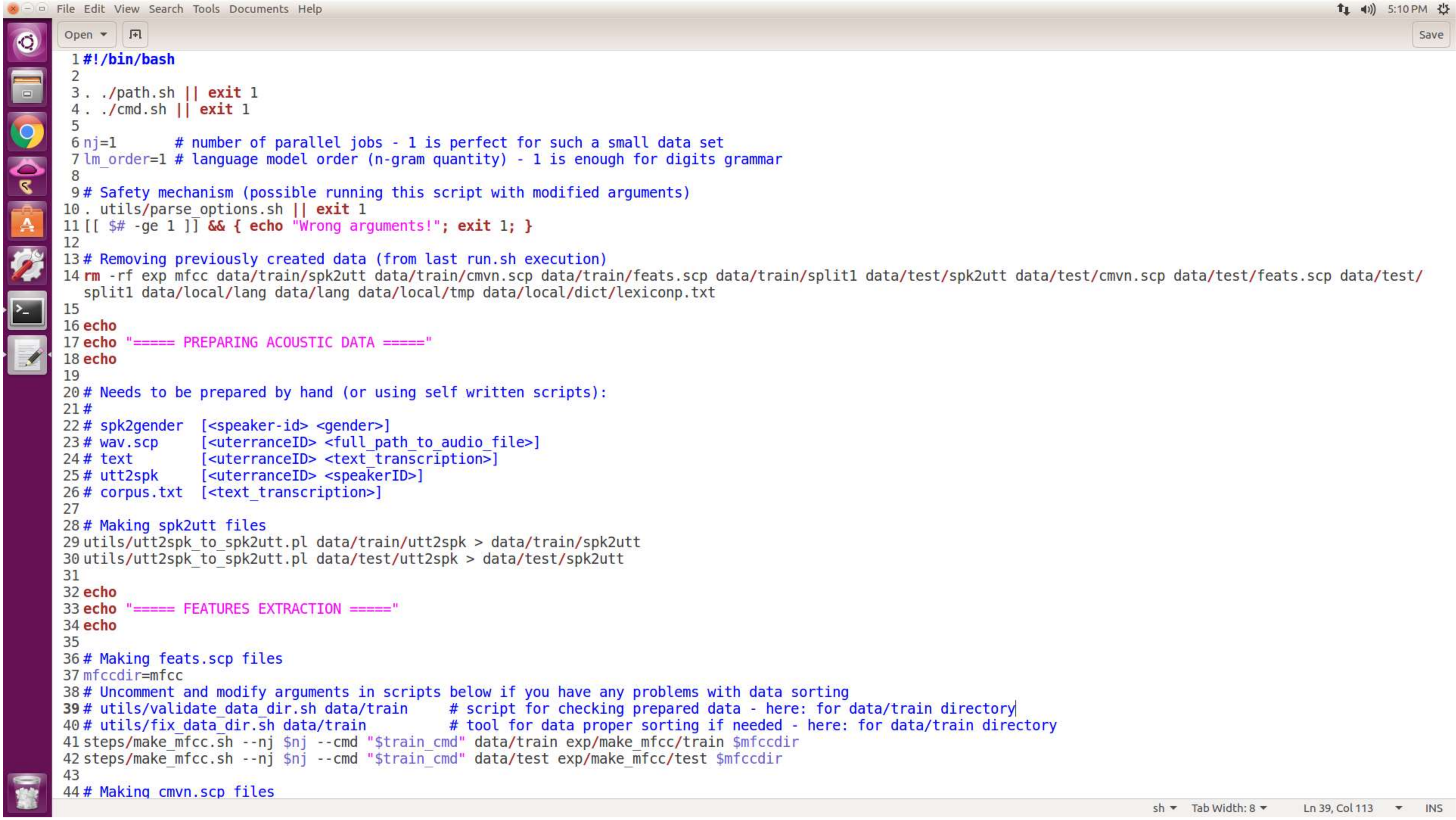
```
# Defining Kaldi root directory
export KALDI_ROOT=`pwd`/../..

# 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/s
gmm2bin/:$KALDI_ROOT/src/fgmmbin/:$KALDI_ROOT/src/latbin/:$PWD:$PATH

# Defining audio data directory (modify it for your installation directory!)
export DATA_ROOT="/home/csce666/Desktop/kaldi/egs/digits/digits_audio"

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

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

Modify to match your setup

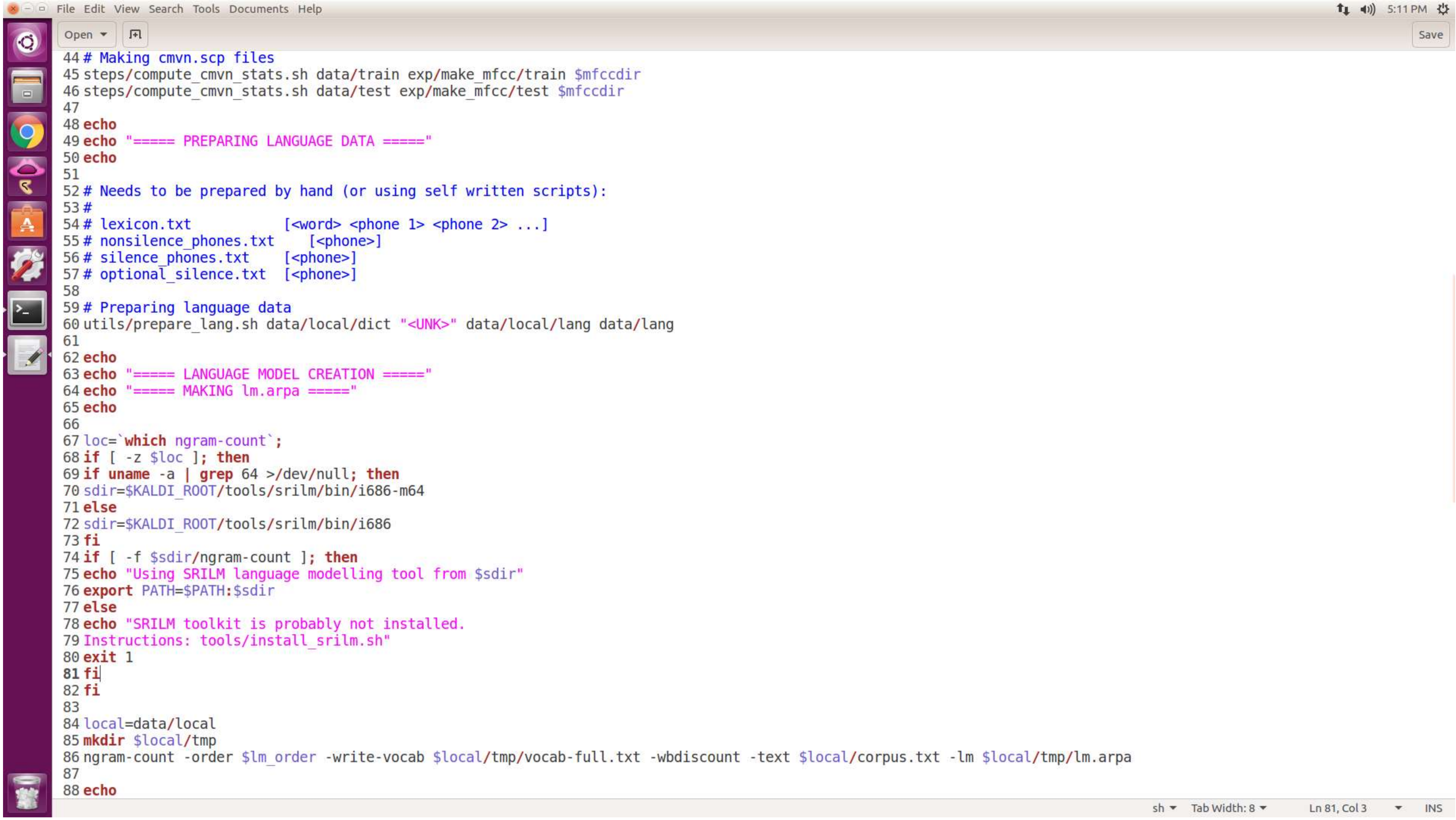
run.sh

```
# Too long to include here. See http://kaldi-asr.org/doc/kaldi_for_dummies.html
```

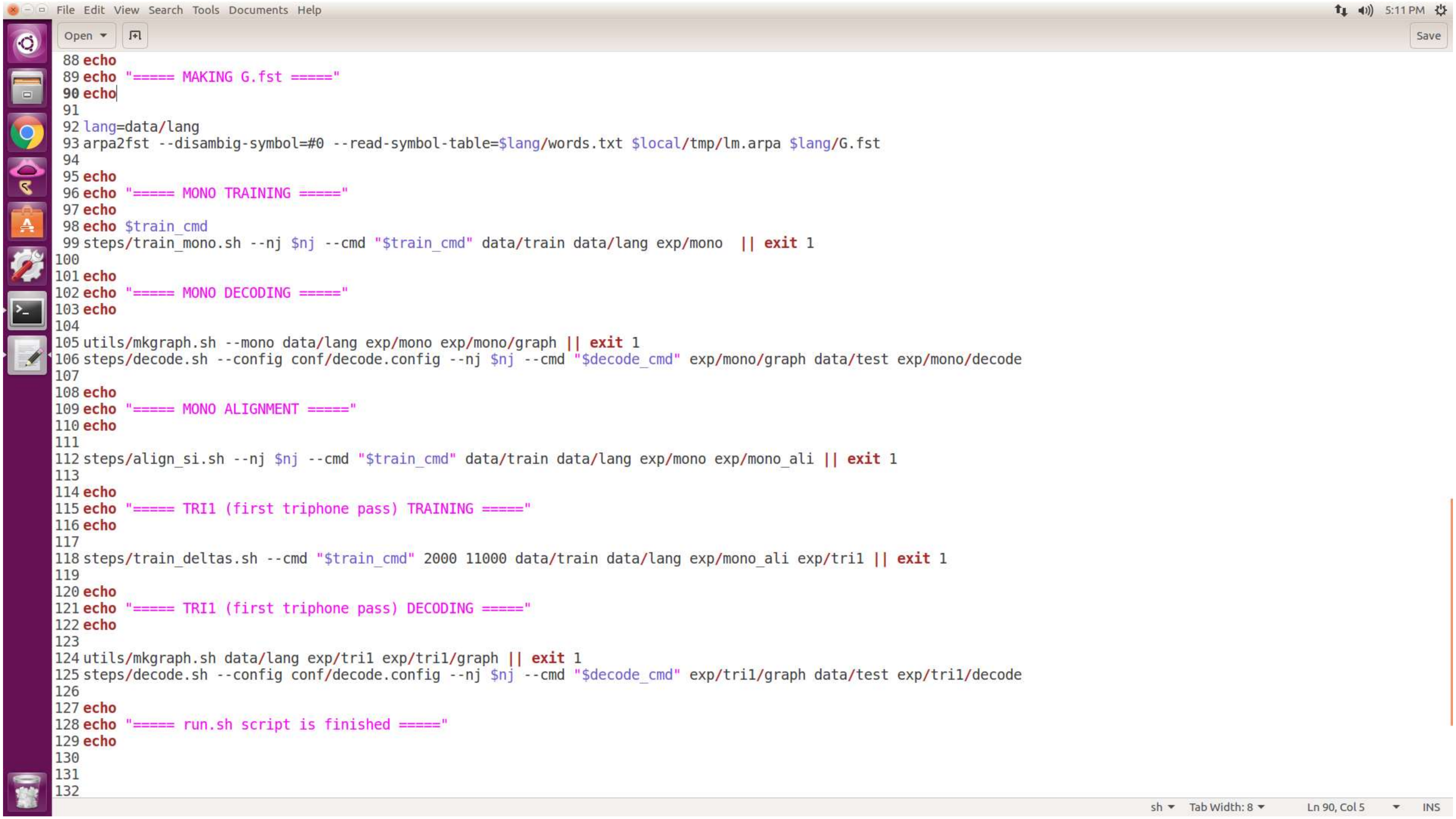
run.sh



```
#!/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
```



```sh
# 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
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
echo $train_cmd
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



```