Documentation Kaldi Speech Recognition

Why Kaldi

* Open source
* Best WER for open source speech recognition frameworks (big competitor is CMU-Sphinx)
* Google and Microsoft also have good models, but those are not free
* Kaldi has a long and very detailed documentation but is way harder to use than the models of the big companies
* Documentation at: http://kaldi-asr.org/doc/ (super big but definitely helps a lot)
  + most important chapters from the docs:
  + <http://kaldi-asr.org/doc/kaldi_for_dummies.html> – good introduction which shows the most important aspects of Kaldi
  + <http://kaldi-asr.org/doc/data_prep.html> – Needed for Data preparation, even though a lot of that is described here
  + <http://kaldi-asr.org/doc/tutorial_running.html>-

How speech recognition works with Kaldi

* Kaldi is very low-level and so you will probably get in touch with phonemes, frames and all the speech-relevant features very quickly anyway
* I built higher-level interfaces that are documented later under the points “Train a model” and “Transcribing your own audio data with a built model”
* This chapter should be a small, very rudimentary introduction to speech-recognition
* Speech consists of words and words consist of phonemes (the sounds of letters basically)
* Some phonetic representations of words:

Abendrot '?a:-b@nt-Ro:t

Abends '?a:-b@nts

Abendstunden '?a:-b@nt-StUn-d@n

* **A model in Kaldi basically consists of 3 different parts:**
* **1.: the acoustic model (the audio/wav-files)**
* **2.: the language model (the transcriptions of the audio-files)**
* **3.: a dictionary which contains exactly those phonetic representations of words**
* For the acoustic model a speech corpus with transcriptions is needed  I used the one from the TU Darmstadt (TUDA) (<http://ltdata1.informatik.uni-hamburg.de/kaldi_tuda_de/german-speechdata-package-v2.tar.gz>) (16 GB, 36h) and the TIB-AV-corpus (16h)
* Kaldi is able to train Gaussian-Mixture-Models (GMMs) in combination with Hidden-Markov-Models and also DNNs instead of the GMMs
* A very short and good explanation of GMM-HMMs can be found here: <https://www.quora.com/How-does-GMM-HMM-model-work-in-ASR>
* I yet only trained GMM-HMMs
* For the TUDA-example I tried training the DNN with the given local/run\_dnn.sh-script and also with the scripts in local/online which both didn't work out.
* Anyway the results are not probable to be a lot better (for the tuda-speech-corpus the GMM-HMM-model gets a WER of 20 % while the WER of the DNN is 19%)
* Set up of Kaldi is automated in the Dockerfile
* Kaldi folder structure (src, tools, egs)
  + Egs is originally the most important folder (contains the examples where the models are located)
  + Every folder in egs-dir is for a model
  + a model basically consists of a exp-folder and a train-folder – some can be downloaded from here: (maybe one day a very good German model as well: http://kaldi-asr.org/models.html)
  + **IMPORTANT: we don’t use the egs-folder anymore since we use docker and would hence lose our models because we decode on a different container than we train.**

** we save our models to the volume that we mount into the docker-container and save it in the /data/models directory**

* + every model contains an s5-directory in which all parts of the model are located
  + Cmd.sh contains the configuration (how many cores are used for training and decoding  since the numbers already differed in the example of TUDA (16 and 5) I decided to also use more cores for the training than for the testing) (30 : 10)
  + Path.sh includes the kaldi dir
  + Most important: run.sh  is the script that finally builds the model
  +  this script is later run by the train.sh-command that you run when starting the docker-container
  + It also downloads the language model and dictionary so if you want to make changes to these parts you need to fork my skeleton-repo (<https://github.com/JuliusCosmoRomeo/kaldi-tuda-de>) and adapt the run.sh-file

What the Dockerfile does

We prepared a Dockerfile (can be found here https://github.com/JuliusCosmoRomeo/kaldi\_interface)

The Dockerfile prepares the Kaldi-setup, downloads Marytts (described later under the chapter “Mary”) and the Kaldi Interface that I developed for easier training and decoding.

At the end of the Docker build the docker-image internally looks as follows (the most important directories are listed):

/opt/kaldi – contains the Kaldi setup

/opt/mary/marytts-5.1.1/marytts-5.1.1 – contains the Mary TTS setup of the specific version 5.1.1

/kaldi\_interface – contains the train.sh and the decode.sh that can be run via docker run

/data – this directory needs to be mounted into the docker-container. The complete structure of how the data-directory on the Host-system should look like can be found in the chapter Train a model – How the dir-structure of the mounted directory should look like

Train a model

* First you need data, that you then mount into the docker container (docker run -v %volume\_path%:/data)
* For some data sources the data are already prepared (TIB-AV-dataset and TU Darmstadt-dataset)  if you want to use these data you can skip the data-preparation-part in the following → the data are located in the data/train\_data -dir
* when you want to train on these data don't forget to pass the particular train\_data-folder (in which the train/, test/ and dev/ folders are located) to the train.sh-command

Data preparation:

* Everything is done in utterances which are ca 5 seconds long
* One utterance for training consists of two important parts: a wav-file of the audio utterance + an xml-transcription-file (exact format is following in the next section)

How to get the audio data:

* Get audio data (if from video data, then you can modify and use /data/video-to-wav-converter.py and exchange the directory in line os.chdir(“/data”) on top of the script)
* Audio must be pcm-encoded with 16k sample rate and mono
* Usually use this command to convert videos or audio into correct format
* ffmpeg -i video\_name.mp4 -acodec pcm\_s16le -ss 00:00:05 -t 00:00:20 -ac 1 -ar 16000 video\_name.wav
* -ss = start time stamp
* -t = duration

How the XML-files must look like:

* THE XML TRANSCRIPTIONS MUST BE IN THE FOLLOWING FORMAT:
* Must include all the tags

*<?xml version='1.0' encoding='utf-8'?>*

*<recording>*

*<rate>16000</rate>*

*<gender>male</gender>*

*<angle>0</angle>*

*<bundesland>Berlin/Brandenburg</bundesland>*

*<muttersprachler>Ja</muttersprachler>*

*<corpus>Tib-AV</corpus>*

*<sentence>Verformung durch die Probe hindurch.</sentence>*

*<cleaned\_sentence>Verformung durch die Probe hindurch.</cleaned\_sentence>*

*<ageclass>18-24</ageclass>*

*<speaker\_id>12671\_76</speaker\_id>*

*</recording>*

* For TIB-AV-transcriptions I wrote a script that extracts the important information from the given XML and adds the other necessary infos to the new XMLs (/data/tools\_for\_data\_prep/transcription-formatter.py)  puts all the new xmls into phrases folder

How the dir-structure of the mounted directory should look like

* On your Host-system you should have a volume named data that you mount into the top-level-directory of your docker-container (via the -v-option of docker (docker run -v ))
* inside the data-directory you should have a “models”-folder
* This folder should contain a folder for each model you want to train or already trained
* In these folders you have a wav-folder for the audio/transcription data and a s5-folder in which the training-scripts are located
* Split the audio-xml-data into three sub-directories within the wav-dir where each dir contains roughly the following percentages of the overall data (train ~ 86%, test = 7%, dev = 7%)
* The wav-file must be in the same dir as the xml-file

 The structure of the data folder on the host-system and hence on the docker container should look as follows:

The red folders are added when running the training-script

------ data/

------------ models/

------------------ model1/

------------------------ wav/

------------------------------ train/ (contains wav-/xml-files: e.g. 10455.wav & 10455.xml)

------------------------------ test/ (contains wav-/xml-files)

------------------------------ dev/ (contains wav-/xml-files)

------------------------ s5/ (the cloned baseline-repo)

------------------------------ run.sh

------------------------------ decode.sh

------------------------------ data/

------------------------------ …

------------------ model2/

------------------------ wav/

------------------------------ train/ (contains wav-/xml-files)

------------------------------ test/ (contains wav-/xml-files)

------------------------------ dev/ (contains wav-/xml-files)

------------ decoding\_data/

------------------------ decoding\_set1/ (contains some wav-files)

------------------------ decoding\_set2/ (contains some wav-files)

------------ train\_data/

------------------------ tib\_av\_data/ (contains dev, test and train dirs)

------------------------ tuda\_speech\_corpus/ (contains a download.sh-script: executed in training)

------------ tools\_for\_data\_prep/

------------ tib\_av\_original\_data/

Running the training-script:

* The following command trains a GMM-HMM-model on your own data or on prepared data

**docker run -v volume\_dir:/data /kaldi\_interface/train.sh model\_name**

or

**docker run -v volume\_dir:/data** **/kaldi\_interface/train.sh model\_name [path\_to\_data\_dir] [utterance-postfix]**

* This command will clone the baseline-model-repo, **link** the data to the data-directory within the new model and then run the run.sh-script
* The first command expects the data (the wav-files and the xml-transcriptions) in folders within /data/models/model\_name/wav
* if you want to train models on the data of the TU Darmstadt the data, use
* I left you the downloaded data from TU Darmstadt within the directory /data/german-speechdata-package-v2
* For these data we have different microphone data so you need to specify an utterance-postfix for the microphone that you want to use
* available utterance-postfixes are:
* \_Kinect-Beam
* \_Kinect-RAW
* \_Samson
* \_Yamaha
* \_Realtek

Mary

* When training the model you need to have the Mary Text-To-Speech server running
* It is started within the train.sh and the decode.sh-scripts with nohup and will be killed when the process stops
* the command to start mary is nohup /opt/mary/marytts-5.1.1/marytts-5.1.1/bin/marytts-server &
* **MaryTTS is used to generate phoneme entries in the phoneme dictionary for words** **that are not in the dictionary yet (OOV-words = out of vector)**
* Further documentation on why exactly MaryTTS is needed can be found here: https://github.com/tudarmstadt-lt/kaldi-tuda-de
* Not having Mary running also results in the following error:
* HTTPConnectionPool(host='127.0.0.1', port=59125): Max retries exceeded with url: /process (Caused by NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object at 0x7f2e8ccc2210>: Failed to establish a new connection: [Errno 111] Connection refused',))
* ERROR:root:
* Traceback (most recent call last):
* File "local/data\_prepare.py", line 207, in getUtterances
* clean\_sentence\_tokens,token\_phonemes = common\_utils.getCleanTokensAndPhonemes(cleaned\_sentence,mary)

Transcribing your own audio data with a built model

Data preparation:

* Test data should also be in the same wav-format as for training purposes  this command might help again:

ffmpeg -i video\_name.mp4 -acodec pcm\_s16le -ss 00:00:05 -t 00:00:20 -ac 1 -ar 16000 video\_name.wav

* Furthermore kaldi needs 3 more files in the directory (taken from kaldi-asr.org/doc/data\_prep.html) that are all already generated by my decode.sh-script
* **Most important (and only one you could adapt: the “text”-file)**
* A very detailed description about what these files are needed for is available under the link above and in short below

1. The text file

* As mentioned above all files are automatically generated, anyway the “text”-file is the only one which you may want to add manually to the folder of your test-data.
* contains the transcriptions of each utterance (the **ground truth**)
*  Makes WER-calculation possible.
* Kaldi-scripts will compare the output of the decoding with the ground truth-transcriptions in the “text”-file
* In the automatically generated file the transcriptions are left “blank” (therefore you have to write “None” behind the id)
* As Id we use the name of the wav-file

s5# head -3 data/train/text

10455 Die Windmühle in Moorsee symbolisiert den technischen Wandel. Im Jahre 1840 erbaut, betrieb man sie ….

10630 None

2. The wav.scp file (automatically generated)

- maps id to file-location. Location can be relative.

- The format of this file is

<recording-id> <extended-filename>

Example: if your folder-name is tibAvTestSet this folder will then be copied into the data-directory of the model  the path you should note in the wav.scp must look as follows:

10455 data/tibAvTestSet/10455.wav

10630 data/tibAvTestSet/10630.wav

3. The utt2spk file

- This file maps utterances to speakers. Since we don’t need speaker-information we just copy the uuid of the utterance as speaker-id

10455 10455

10630 10630

Running the decoding-script:

**docker run –v volume\_dir:/data /kaldi\_interface/decode.sh modelname absolute-path-to-test-dir**

The command copies the data to the data dir of the model and then runs the decode.sh-script within the model-directory

Some trained models are already available in the /data/models-directory

These are:

* tuda\_samson
* tuda-kinect
* tuda-yamaha
* tuda-kinect-raw
* tib-av-model

Already built models and their accuracy on TIB-AV-data

Yet I mainly trained models with the already existing recipes from TU Darmstadt (<https://github.com/tudarmstadt-lt/kaldi-tuda-de>) and an own model built from the TIB-AV-data

These models are all located in the /data/models directory

My Baseline-model (also used in the train.sh-script) for training also builds on this repo and is located at <https://github.com/JuliusCosmoRomeo/kaldi-tuda-de>

With the TUDA-daat multiple models can be built because data from multiple microphones are available

Good results (WER ~20 %) when tested on the same kind of data (from the same microphone)

Means: when using e.g. the Kinect-data for training you will only have good decoding-results for Kinect-test-data (source: Benjamin Milde, co-author of the TUDA-model-recipe in Kaldi)

→ Problem: not really good when data comes from different microphones

In the TIB-AV-corpus the train-data come from many different microphones

Results for TIB-AV-test-dataset of 4 Videos (~78 mins of audio)

Samson: 73,77% WER

Tib-AV-model: 87,22% WER

Yamaha: 78,43% WER

Kinect RAW: 75,91% WER

Kinect Beam: 81,31% WER

* No model actually gave us very satisfying results, from the yet trained models the Samson-model was the best

Future work

Future ideas include combinating the best TU-Darmstadt-models and the TIB-AV-model to train the model on a bigger variety of speakers.

Therefore it is important that all wav-files for training are in the same format: avoid utterance-postfixes for the TUDA-data

The IDs don't need to be in a specific format. It is only important that a xml-file exists for every wav-file.

Furthermore I guess that the dictionary limits the accuracy of the models because many words from the TIB-AV-portal are not included in the phonetic dictionary and Mary probably does not get the phoneme entries for every OOV-word right. → Maybe we can find a better phoneme dictionary?

Maybe also changes to the language model could improve the results. Yet we only used the language model with its texts from TUDA and only trained on different acoustic models.

For both the language model and the dictionary you should adapt the run.sh-script in the <https://github.com/JuliusCosmoRomeo/kaldi-tuda-de> repo in the s5-folder.