KTH Computer Science and Communication

DT2118 Lab3A: Continuous Speech Recognition

1 Objective

The objective is use the Hidden Markov Model Toolkit (HTK) to train and test Automatic Speech Recognition models based on Hidden Markov Models and Gaussian Mixture Models. The focus is on analysing the effect on training on the model parameters, and test different feature extraction methods. All the methods are implemented by HTK.

2 Task

Train and test a continuous speech recogniser for digits.

- use the provided scripts to prepare the data
- describe the properties of the data observing the information generated by the previous step
- use the provided scripts to train and test Gaussian HMMs
- analyse the effect of different feature kinds
- use the provided scripts to train and test Gaussian Mixture Models HMMs
- analyse the effect of increasing the number of Gaussian components
- tune the word insertion penalty parameter
- analyse the results of forced alignment
- describe the evolution of the parameters at different training iterations
- describe the difference between models for different phonetic classes.

In order to pass the lab, you will need to follow the steps described in this document, and produce a report where you describe your work and answer the questions asked here. The report should be submitted in the Assignment section in the course Web on KTH Social `https://www.kth.se/social/course/DT2118/`. One submission should be done for each group, clearly stating all the members of that group.

3 Preliminaries

The training is based on a number of shell scripts, some Python scripts and the HTK toolkit[1]. In order to run the training you will need to setup your environment so that all the tools are available. This requires, for example installing HTK and, if you use MS Windows, possibly installing Cygwin[2]. If you want to avoid wasting time with these practical issues, I advice you to run the lab on one of the Ubuntu machines at CSC. On those computers all the required software is already installed and configured. The lab should work on Mac OSX as well, but has not been thoroughly tested.

The lab directory contains the following directories:

- `config`: configuration files
- `tools`: scripts for training and testing
- `htkModelParser`: a Python class to parse HMM definition files in HTK format (needed at the end of the lab)

And the following files:

- `lab3a.pdf`: this file
- `htkbook.pdf`: the HTK manual also called KTH Book

If you use an Ubuntu machine at CSC, run the command

```
source tools/modules
```

If you are at one of the Mac OSX machines, run instead

```
export PATH=/afs/nada.kth.se/dept/tmh/hacks/pkg/htk/3.4.1/os/bin:$PATH
```

> The above commands add the necessary information to the search path and have to be run every time a new terminal is opened.

3.1 Data

The speech data used in this lab is from the full TIDIGIT database (the data for Lab 1 and Lab 2 was a small subset of the data used here). The database is stored on the AFS cell `nada.kth.se` at the following path:

```
/afs/nada.kth.se/dept/tmh/corpora/tidigits
```

If you have continuous access to AFS during the lab, for example if you use a CSC Ubuntu machine, create a symbolic link in the lab directory with the command:

```
ln -s /afs/nada.kth.se/dept/tmh/corpora/tidigits
```

Otherwise, copy the data into a directory called `tidigits` in the lab directory, but be aware of the fact that the database is covered by copyright[3].

[1] `http://htk.eng.cam.ac.uk/`
[2] `https://www.cygwin.com/`
[3] See `https://catalog.ldc.upenn.edu/LDC93S10` for more information.

Script	model kind	task	comments	time
`prepare_data.sh`	-	preparation	prepares the data (uses additional Python scripts in `tools`)	∼2s
`train_g-hmm.sh`	GHMM	training	performs flat initialisation and Baum-Welch training	∼6m
`test_g-hmm.sh`	GHMM	testing	performs Viterbi decoding and evaluation	∼2m
`train_gmm-hmm.sh`	GMM-HMM	training	increases the number of Gaussian components per state to 2, 4, 8, and 16 and retrains	∼11m
`test_gmm-hmm.sh`	GMM-HMM	testing	evaluates the models from `train_gmm-hmm.sh`	∼12m
`forced_align.sh`	GMM-HMM	forced alignment	performs forced alignment on the training data using GMM-HMM	∼1m

Table 1. Main training and evaluation scripts. Times obtained with a single core i5 CPU at 2.5GHz.

4 Training and Testing

There are six main Bash scripts to train and test the recognition models under the directory `tools`. Table 1 summarises them. You will use them to test the effect of using different features sets and increasing the number of Gaussian components for each state in the model set.

4.1 Prepare the database

Run the data preparation script with

```
tools/prepare_data.sh
```

This will create a `workdir` directory, with a number of files containing information about the database. All files are text based. Open the files with a text editor and observe their content (take advantage of the descriptions given by the script).

Report the following information:

- number of male and female speakers in the training and test set respectively
- number of training and test utterances
- number of phonemes (including silence)
- number of nodes and arcs in the recognition network (`workdir/digitloop.lat`[4])
- Optional: display the content of the grammar in `workdir/digitloop.lat` as a graph (requires graph producing software as graphviz[5] and some scripting to convert between file formats).

[4] Refer to Section 12.2 in the HTK Book for information about the HTK Standard Lattice Format
[5] `http://www.graphviz.org/`

4.2 Train and test G-HMMs with different features

Train Gaussian HMMs (G-HMMs) for different feature kinds listed in the following table:

Feature code	Description
`MFCC`	Mel Cepstrum Coefficients
`MFCC_0`	plus zeroth coefficient
`MFCC_0_D`	plus deltas
`MFCC_0_D_A`	plus acceleration
`MFCC_0_D_A_Z`	with cepstral mean subtraction

For each feature kind, a feature extraction configuration file is provided in

`config/features_<feature_code>.cfg`

and a HMM definition prototype is provided in

`config/proto_<feature_code>.cfg`

Check the difference between these files and, in case refer to the HTK Book to understand their meaning[6]. Report the number of features for each feature kind and the number of parameters for each prototype model (number of states and number of free parameters).

Repeat the following steps changing the feature kind each time.

> Note from Table 1 that depending on the speed of your computer, the whole training and testing for G-HMMs for one feature kind can take up to 10 minutes. On multi-core computers, to increase the speed, run the different features in parallel by starting as many terminals as the number of feature kinds (but at most as many as the number of cores). On linux systems you can check the number of cores with
>
> `cat /proc/cpuinfo | grep processor`

Set the feature kind with:

`features=MFCC`

Now train the G-HMM models with:

`tools/train_g-hmm.sh $features`

The script runs flat initialisation (global means and variances for all states) with `HCompV` and then Baum-Welch reestimation with `HERest`. After some iterations, a short pause model (`sp`) is added with `HHEd` to model possible silence between words in the same utterance.

The models are be stored in the directory `models_$features`. Check the training script for details. Explain the difference between models stored in `hmm1-3` and `hmm5-7`. Also, explain the difference between the models in `hmm4` and the ones in `hmm5`[7].

Evaluate the models by running

`tools/test_g-hmm.sh $features`

The script runs the Viterbi decoder (`HVite`) and summarises the results (`HResults`) for the test utterances. The results are both displayed in the terminal and stored in the file

`results_$features/results_g-hmm.txt`

Refer to the HTK manual, `HResults` reference section, to interpret these results. Report and comment the word accuracy and correct words for different feature kinds.

[6] Relevant parts are Chapter 5 for feature extraction and Chapter 7 for HMM definition files.
[7] Refer to Section 3.2.2 in the HTK Book

4.3 Train and test GMM-HMMs with increasing number of Gaussians

Here use the `MFCC_0_D_A` features and train GMM-HMM models with 2, 4, 8 and 16 Gaussian components per state. If you want, you can test the same procedure with other feature sets by changing the value of `$features`.

```
features=MFCC_0_D_A
tools/train_gmm-hmm.sh $features
```

Now test the models with

```
tools/test_gmm-hmm.sh $features
```

The results are both displayed in the terminal and stored in the file

```
results_$features/results_gmm-hmm.txt
```

Report and discuss how accuracy and correct words change with increasing number of Gaussian components.

4.4 Tune insertion penalty parameter

`HResults`, that is run by the test scripts, reports, besides the Accuracy, the number of insertions and deletions. Choose one of the experiments (model set and feature kind) and check if the number of insertions and deletions is balanced. In case it is not, try running the recogniser with different settings for the Insertion Penalty parameter. To do this, modify the `test_g-hmm.sh` or `test_gmm-hmm.sh` changing the `-p` option in `HVite`. Report how this parameters affects the results both in terms of insertions and deletions and in terms of accuracy.

4.5 Forced Alignment

Run forced alignment on the training material with the GMM-HMM models with 16 Gaussian components with the following command:

```
tools/forced_align.sh MFCC_0_D_A
```

If you want, you can run the command with other feature kinds instead, provided you have trained GMM-HMM models for those.

Check the alignment between phonemes and sounds using Wavesurfer, following these steps (change the file name at will):

1. run `wavesurfer tidigits/disc_4.1.1/tidigits/train/man/gr/7566116a.wav`
2. in the `Interpret Raw File As` window, choose:
   - `Sampling rate=20000`,
   - `Read Offset (bytes)=1024`,

   click on `OK`
3. in the `Choose Configuration` window, choose `HTK transcription`
4. right click on the transcription pane (the white line starting with ".lab") and choose `Properties...`

5. click `Browse...` in the `Master Label File` field and choose `workdir/train_aligned.mlf`

6. in the `Trans1` tab, change `Label filename extension` from `.lab` to `.rec`, and click `OK`.

Check for example if fricatives such as `s` and `f` are clearly distinguishable in the spectrogram. Also check if the silence at the beginning and end of the utterance is well aligned, and if there is any short pause (`sp`) between consecutive words. Report if you notice any misalignment.

5 Analysis

Here you will analyse some of the models you have trained in the previous section.

5.1 Parameter evolution at different iterations

In `models_MFCC_0/` the `hmm` directories contain models at different iterations. Checking the `train_g-hmm.sh` script, you will notice that G-HMM models (single Gaussian per state) are stored in the directories from `hmm0` to `hmm7`.

Write a Python script that reads the HTK model definition files and displays the evolution of the model parameters for different iterations. In `tools/htkModelParserExample.py` you have an example on how to parse and use the `Hmm` class from `htkModelParser` to parse and use the HTK model definition files.

You do not need to show the evolution of all the parameters in a model set. Rather, choose a phoneme and a state (one of the three) and follow the evolution of means and variances along the iterations. What conclusions can you reach? Do the model parameters converge after seven iterations? Do they converge earlier?

5.2 Parameters for different phonemes

Now use the models in `hmm7` and the middle state for each phoneme model. Check the mean vectors for different phoneme classes: compare fricatives (for example `s`) with vowels (for example `ao`) and with silence (`sil`).

Can you interpret the results? For this, consider that the means correspond to MFCCs.

> Note that HTK, for compatibility with other feature kinds, puts the MFCC coefficient number 0 at the end of the feature vectors instead of the beginning. You will need to swap the position of the 0'th coefficient in order to be able to interpret the results. With numpy arrays this can be done with slicing.

Invert the MFCC computations by running the DCT transform again in order to obtain the log outputs of the filterbank (remember Lab 1). Can you interpret the results now?