

SGN-4507 Speech Recognition Laboratory

Instructions for the Project Work

<http://www.cs.tut.fi/courses/SGN-4507/>

1 General

This course consists of implementing a Finnish word-level digit recognizer with the Cambridge *Hidden Markov Model Toolkit* (HTK). There will be one introductory lecture after which the work will be done independently in groups of 1-2 people. We will get together once more about 2 weeks before the deadline to discuss and find solutions to problems that people have come across (the exact time and date will be put on the webpage of the course).

For the project you will need an account in Birdland (the cs.tut.fi-computer network, <http://www.cs.tut.fi/lintula>). You can use the Linux-machines in classrooms TC415 and TC407. To complete the project, go through the steps given in these instructions and return the required items in a report by email in ASCII,RTF or PDF-format to konsta.koppinen@tut.fi. In addition to returning the report, agree on a short meeting with the lecturer where you demonstrate going through the project work on a computer. The **deadline** for returning the report is **23.5.2010**. The project is graded on a scale of pass/fail.

For completing the project work, you will need to read the first 3 chapters of the *HTK Book* and some sections from the later chapters as well. A link to the book is given on the course webpage

<http://www.cs.tut.fi/courses/SGN-4507/>

A fair amount of text file processing is required. For this, knowledge of a scripting language such as Perl, awk or sed is so useful as to be almost necessary. However, you can use any language of your choice (even Matlab with fscanf and fprintf if you like). It is a good idea to save the commands you give on the command line to a file so you can easily remember/modify them. Most of these you will also need to return in the report.

The speech and other data for processing is located in Birdland in the directory `/share/kurssit/SGN-4507/SpeechDat_digits/` so have a look at it and its subdirectories. The first time you change to the directory, type the full directory into the `cd` command, the shell might not be initially able to complete the directory with TAB.

You should use HTK version 3.4 to do the work. To ensure that you have the correct HTK version in your path, you need to add

```
/usr/local/opt/htk-3.4
```

to your path and ensure that the directory appears before the directory `/usr/local/bin`

in your `PATH` variable¹

The project will be a simplified/modified version of HTK Book chapter 3, where you will find more details on the steps below. The later chapters of the book explain in more depth how the HTK tools work. Any reference to a page number or a section such as 3.2.1 refers to the HTK Book.

2 Creating the dictionary

The wave-files for training the digit recognizer are in

```
/share/kurssit/SGN-4507/SpeechDat_digits/WAV.
```

All of the transcriptions of the files are given in the file **labels.txt** (in the parent directory), take a look at it.

For non-Finns (and the forgetful Finns), the Finnish digits are:

- 0: nolla
- 1: yksi
- 2: kaksi
- 3: kolme
- 4: neljä
- 5: viisi
- 6: kuusi
- 7: seitsemän
- 8: kahdeksan
- 9: yhdeksän

The files in `label.txt` list the files with a `fio`-ending but you can get a corresponding `WAV`-file by simply changing the ending to `wav`.

First create a task grammar for a single digit in the file **gram** as described in HTK Book chapter 3.1.1 except that there are no repetitions of the digits and the digit is not optional. Use `!ENTER` and `!EXIT` instead of `SENT-START` and `SENT-END`. For the digits, use all uppercase characters *with no ä's* (or `å,ö` if they were needed). You can for example use `SEITSEMAN` to denote `SEITSEMÄN` and `seitseman` to denote its transcription. Note that `!ENTER` and `!EXIT` are the first in alphabetical order and they have to be specified an empty output string by writing `[] sil`, similarly to that for `SENT-START` on page 26. After creating `gram`, create the low-level representation with `HParse` into the file **wdnet**.

Next, create a dictionary into the file **dict** which has all of the task words (digits) in uppercase (including `!ENTER` and `!EXIT`) in *alphabetical order*, and the corresponding transcriptions. We will be using word models (not phoneme models) so just use the lowercase version of the digit name. This is described in more detail in Chapter 3.1.2 of the HTK book. Run the command

¹In practice, you can write `PATH=/usr/local/opt/htk-3.4:${PATH}` on the command line or add that line to the file `.bashrc` in your home directory (if you are using bash shell).

`HdMan -n model_list dictionary dict`
to get a list of the models and a slightly modified dictionary-file which will be used later.

Items to Return

Include the following in the report:

- the file `dictionary`
- the file `model_list`

3 Script files

Next we need so-called Master Label Files (MLF) and script files for the speech data. The MLF's will tell the recognizer what words are contained in a given speech file. This information is needed in training. For evaluating the recognizer, we will need this information for the test files as well.

Based on the information in 3.1.4 and the file `labels.txt`, do a program to create a MLF **train.mlf** which contains the transcription of the first 2500 files in `labels.txt`. Be sure to have the words in the same form as they are in `dict`. At this point also create a script file **train.scp** which simply has a list of all the training files with a `mfc`-suffix, i.e.

```
/share/kurssit/SGN-4507/SpeechDat_digits/MFC_Z/a10000i1.mfc  
/share/kurssit/SGN-4507/SpeechDat_digits/MFC_Z/a10001i1.mfc  
/share/kurssit/SGN-4507/SpeechDat_digits/MFC_Z/a10002i1.mfc  
...
```

The `mfc`-files will contain the Mel-frequency cepstral coefficients which we'll do in a while.

Do the same thing for the next 500 files in `labels.txt` to get the files **test.mlf** and **test.scp**. Thus there will be no common samples between the training and testing data.

Next we will need the phone-level MLF-files, which HTK can generate from the word-level MLF and the dictionary. In our case this step is not very interesting (though necessary) since we are using word-level models but it would be important if we built phoneme-level models. Run the `HLEd`-command on page 28 to generate the 'phoneme-level' transcriptions into the file `train_phones0.mlf`. You will need to create the edit script `mkphones0.led` for this. Do the same thing for the test data, i.e. create the phoneme-level MLF-file `test_phones0.mlf`.

Items to Return

Include the following in the report:

- the program you use to create the MLF from `labels.txt`
- the `HLEd` command line for creating the phoneme-level training MLF

4 Feature extraction

Next we will perform feature extraction on the speech files to generate files with the Mel-frequency cepstral coefficients (MFCCs). These take up a fair amount of space and have been precomputed in the directory

```
/share/kurssit/SGN-4507/SpeechDat_digits/MFC_Z,
```

so you just need to run feature extraction on a few files to see how it's done.

The mysterious `_Z` in the directory refers to the cepstral mean having been removed from the files.

HTK does feature extraction with the command `HCopy`. As described in 3.1.5, perform feature extraction on the first 3 files of the training data. Create the file **config** which is otherwise the one given on page 29, except: use subtraction of the cepstral mean with the `_Z`-switch, don't save the MFCC's in compressed format, don't use the CRC, do normalize the energy (since we won't be using live audio for recognition), and add the line `SOURCEFORMAT = WAVE`.

Here we are calculating and storing only the MFCC's and the energy (as given by `MFCC_0_Z`, which is zero and not 'oh') and not the delta- and acceleration-coefficients, which will be computed later on-the-fly as they are needed. This feature vector has 13 coefficients and when appended with the deltas and accelerations, it will have 39 coefficients.

After running the feature extraction with `HCopy`, check that the output files are the same as the corresponding ones in

```
/share/kurssit/SGN-4507/SpeechDat_digits/MFC_Z,
```

e.g. with the Unix command `diff`. You can read the information in a MFCC-file using the command `HList`.

Items to Return

Include:

- the `HCopy`-command and the configuration file for it

5 Initialization of the HMMs

Now it's time to start modeling the data (specifically the feature vectors) using HMMs. First specify the model topology as described in 3.2.1 by creating a prototype HMM in the file **proto**. This will be a left-right HMM with 12 emitting states and a diagonal covariance matrix. Note that for HTK the prototype will actually consist of 14 states since HTK has one initial and one final non-emitting state used essentially for 'gluing' different models together. Also create a prototype for the silence model in the file **proto_sil** which has 3 emitting states.

Next you will initialize the parameters of the models with the global mean and variance of the training data using the `HCompV`-command. For this, create a new configuration file called **config_hcompv** which is the same as `config` except for adding `TARGETKIND = MFCC_0_D_A_Z` and removing `SOURCEFORMAT`. The effect of `MFCC_0_D_A_Z` will be to add the delta- and acceleration coefficients to the static MFCC features (as well as removing the cepstral mean), which will result in a feature vector of length 39. As described in 3.2.1, create a new directory `hmm0` where a new version (with updated means and covariances) will be stored.

In general with HTK, updated parameters are stored in a new directory (you'll have about 30 HMM-directories by the time you are done). The HMMs take up a fair amount of space so you may need to delete hmm-directories which you no longer need to stay within your disk quota.

After calculating the global mean and variance, create the master macro file (MMF) `hmmdefs` in the directory `hmm0` which will contain the definitions of all the word models and the silence model. The HTK Book says to create this manually but people with self-respect will write a program for creating the MMF using `model_list`, `proto` and `proto_sil`. Also create the MMF macros as specified in the HTK Book.

Items to Return

Include:

- the files `proto` and `proto_sil`
- the full `HCompV`-command

6 Training and Evaluation of the HMMs

Now that the HMMs are initialized, it's time to start training them. This will be done using the Baum-Welch–reestimation algorithm (also known as the expectation-maximization or EM algorithm) which modifies the parameters of the HMMs based on training data so that the likelihood of the training data is increased.

The training is done with the `HERest`-command, which takes in HMM definitions, a list of training files, the correct model-level transcriptions in a MLF, a set of models to update, and outputs a set of modified HMMs. Do the reestimation 5 times, each time writing the updated HMMs into a new directory. Use the pruning thresholds given in 3.2.1.

After each iteration, do recognition of the *testing* data with the updated models using the `HVite`-command as described in 3.4.1. In addition to the parameters given in the HTK Book, you need to specify the configuration file by adding `-C config_hcompv`. Evaluate the recognition results with the `HResults`-command which gives you the percentage of digits recognized correctly. For this you will need the `test.mlf` (created earlier), model-level transcriptions of the test data in `test_phones0.mlf` (also created earlier) and the file `model_list` which has the list of digits as they are in the dictionary.

Items to Return

Include:

- The full `HERest`, `HVite` and `HResults`-commands for the first iteration
- the recognition results (given by `HResults`) after each iteration

7 Fixing the Silence Model

After the previous training, add transitions to the silence model as described in 3.2.2 to make it fully connected (since noise generally doesn't necessarily have a left-to-right structure unlike speech). This is done using the HHEd-command with a script file `sil.hed`. We won't be using the short pause (`sp`) model so don't specify that.

After modifying the silence model, reestimate the models 5 times and evaluate the performance after each iteration.

Items to Return

Include:

- The HHEd-command and the file `sil.hed`
- the recognition results after each iteration

8 Adding Mixtures

The next step is to increase the amount of mixtures in the HMM states. This is done using the HHEd-command as described in 10.6. Increase the amount of mixtures in each state in each model to 3 and perform reestimation and recognition (with the test data as always) 5 times.

After this, increase the number of mixture components to 5, and again reestimate and evaluate the performance 5 times. Increment once more the number of mixtures to 7 and reestimate and evaluate 5 times. You will get 2637-warnings in HHEd which indicate that some mixture components are estimated with too little training data but just ignore these.

After the final reestimations, the recognition rate for the test data should now be over 90%.

Items to Return

Include:

- The HHEd-command for incrementing the mixtures to 3 and the corresponding script file
- The recognition results after each iteration