Practical exercise in speech recognition 2007

Objective

The objective is to get in contact with methods for training, recognition and evaluation by using a common software package (HTK) for designing a simple speech recognition application.
The HTK package is available for free download at http://htk.eng.cam.ac.uk/ . Its manual, the HTK Book, can be downloaded separately.

Task

Specify a simple application on recognition of questions regarding GSLT students travelling. Topics are countries, cities, flights, hotels, etc. Design a grammar and a vocabulary. Record training and test data. Train monophone, single-mixture models. Evaluate the recognition performance on the test data. The exercise follows closely the Tutorial Example, steps 1-8 and 11 in the HTK Book.

Besides the grammar design and the speech recording, a major part of the exercise consists of executing a number of scripts and HTK programs. To make it possible to complete the exercise in the limited time available, the required commands have been prepared in this document and also in a corresponding text file (`htk_lab/Practical_commands.txt`) for easy cut-and-paste operation. Although the commands can be performed in an automatic fashion without understanding, it is highly recommended to consider the function of the individual steps in the process.

Optional exercise extension

The exercise can be extended into a term paper for the closing seminar. The following topics are possible. It should be noted that the available training corpus is too small for detailed acoustic and linguistic modeling.
Phoneme recognition on a larger corpus, e.g. TIMIT
Adaptation
Other spectral features
Speaker recognition
Other grammars
Speaker dependence

Prerequisites

Basic familiarity with Linux and emacs editor.

Project directory

The students will work in groups of 2 or 3 in individual user directories within a common project root directory. The root directory contains scripts, dictionaries and other tools.

Directory structure:

```
htk_lab/
   Dictionary/
              beep/
```

```
Group1/
      Sp1/
            train_data/
                  MFCC/
            test_data/
                  MFCC/
            hmm0/
            hmm1/
            .
            hmm7/
      Sp2/
      Sp3/
Group2/
.
Group6/
```

The root directory htk_lab/ contains a dictionary directory, directories for each exercise group (Group1,...Group6) and common scripts and text files.
The dictionary directory contains the British-English BEEP pronunciation dictionary.
Within each group directory, there are directories for each group member (Sp1, Sp2, Sp3). These contain directories for storing training and test utterances and trained models in various stages (hmm0 ... hmm7).

The exercise follows the tutorial example in the HTK Book 3.3, chapter 3
That manual is stored as `htkbook.pdf` in the htk_lab directory.

Procedure

In the following, HTK and other commands are preceded by the prompt '>'
These can be pasted into the command window by selecting them and clicking the mouse wheel (button 2).

0. PREPARATORY
Change the working directory to Group$m ($m = your group number 1 … 6)
```
>cd /afs/nada.kth.se/dept/tmh/hacks/sandbox/htk_lab/Group$m
```

Open a Terminal window (Tool Bar: Applications -> System Tools -> Terminal)
In that window, open an X terminal window and continue working from there
```
>xterm &
```

Make the programs in the htk package executable by using the Linux `module` command
```
../.modules
```

Open the txt version of this document in a text editor, e.g. emacs, in order to copy the commands
```
>emacs ../Practical_commands.txt &
```

Change to a short command line prompt
```
>set prompt = "%m:%c3>"
```

Set the microphone amplification by Applications ->Sound and Video->Volume Control.
The controls Master, Mic and Mic Boost should be set to: Mute = False, Rec = True

The Capture slide setting is machine dependent:

ctt09	Capture 50%
ctt19:	Capture 75%
ctt15:	No Capture control, Left mic: Max, Right mic: Mute
ctt03:	Capture 50%
ctt41:	Capture 50%
melquiades:	Capture 0%
ctt48 (piglet)	Capture 40%
ctt45	

Copy the configuration file from the mother directory
```
>cp ../config.cfg .
```

1. PREPARING THE TASK GRAMMAR

Copy a grammar file with words and categories defined but with no syntax
```
>cp ../NATIS_no_syntax.grm ./gram
```

Use a text editor (emacs, or possibly Open Office) to insert the syntax
```
>emacs gram &
```

Define a small finite-state grammar of a travel information service application. The grammar should (at least) allow the following sentence types :
"Show <airline> flights from <place> to <place> on <weekday> <part-of-day>"
"When does the <vehicle> from <place> arrive?"
"When does the <vehicle> to <place> leave?"
"Are there any flights to <place> before/around/after <hour><quarter>?"
where <place> is one of the Nordic/Baltic countries and a city in each country
<part-of-day> is one of "morning, noon, afternoon, evening, night"
<vehicle> is one of "flight, plane, boat, ferry, bus, train"
<airline> is one of around 5 airlines

All words should be in upper case.
The HTK Tutorial has an example of a similar grammar.

Extract a word list file 'wlist' from the grammar file
```
>../gram2wlist < gram >! wlist
```

Generate a word network 'wdnet'
```
>HParse gram wdnet
```

2. THE DICTIONARY
Create a pronunciation dictionary for the application. The British English dictionary BEEP has been downloaded and contains around 237 000 words. Words not covered in BEEP, e.g. names, have to be transcribed manually in a separate file. The phoneme inventory used in BEEP is ARPAbet, which is described in the file ../Dictionary/beep/phoncode.doc.

If words are missing in BEEP, they will have to be inserted in a separate dictionary file, ‘names’. An initial version can be copied from the mother directory.

```
>cp ../names .
```

Create a list of phones in file monophones1

```
>HDMan -m -T 2 -w wlist -n monophones1 -l dlog tmp_dict
../Dictionary/beep/beep-1.0 names ../endnames | grep missing
```

The dictionary `'endnames'` contains labels SENT-START and SENT-END for sentence-initial and -final silence. These have no output symbol, and this is specified by empty square brackets in the dictionary file. Evidently, the brackets are lost in the HDMan process and have to be re-inserted in the assembled dictionary file dict.

```
>cat tmp_dict | sed s'/SENT-END/SENT-END \[\]/' | sed s'/SENT-
START/SENT-START \[\]/' >! dict
```

If words are missing, insert these manually in the dictionary 'names' in alphabetic order and repeat the command. If remaining errors, edit the output dictionary 'dict' by hand. The phonetic inventory is given in Dictionary/beep/phoncode.doc, 1st column (ARPAbet)

Print the vocabulary size and the grammar perplexity (quite small)

```
>HSGen -q –s wdnet dict
```

3. RECORDING THE DATA

Repeat for each group member Sp$n ($n = 1,2,3; Sp1, Sp2, Sp3)

```
>set n = 1
```
(and n = 2 , 3)

Generate 30 train and 15 test prompts. Each invokation produces a new random list.
The first item in each line of the prompt files is changed from <line#-full_stop> into <T[S|R]_line#>

```
 >HSGen -l -n 30 wdnet dict | sed 's/.*\./\*\/TR&/' | sed
's/\.//'  | sed 's/SENT-START//' | sed s'/SENT-END//' >!
Sp$n/trainprompts
 >HSGen -l -n 15 wdnet dict | sed 's/.*\./\*\/TS&/' | sed
's/\.//'  | sed 's/SENT-START//' | sed s'/SENT-END//' >!
Sp$n/testprompts
```

Then start the recording program HSLab and the prompting script for the training utterances.

```
>../record_list Sp$n/train_data < Sp$n/trainprompts
```

The prompt text is displayed in the command window (which will probably have to be moved not to be hidden by the HSLab window).
An utterance recording is initialised by clicking the "rec" button
and stopped by clicking the same button (relabeled into "stop").
Microphone position: at the corner of the mouth. Distance: around 2 cm.
The waveform is displayed. The max amplitude should be around 50% of the window height. If any error in an utterance, repeat the "rec"-"stop" sequence as many times as required.
Accept the recording by clicking the "Quit" button, which closes the HSLab window.
The next prompt is then displayed and a new HSLab window is opened.
The recording is repeated until the end of the prompt list is reached.
Unfortunately, only HTK format for the sampled files was possible to use.
Move the xterm window to a position where the prompt text is not hidden by the HSLab window.

Then do the same procedure for the test utterances

```
 >../record_list Sp$n/test_data < Sp$n/testprompts
```

The script also generates a file containing a list of the files recorded.
This file, "files.scp" is written into the Sp$n directory.

4. CREATING THE TRANSCRIPTION FILES

In each speaker directory Sp$n ($n = 1, 2, 3) do the following:

Create word level master label files (name extension .mlf) by using the script prompts2mlf. Syntax: prompts2mlf mlf-file prompt-file

```
>../prompts2mlf Sp$n/trainprompts.mlf Sp$n/trainprompts
>../prompts2mlf Sp$n/testprompts.mlf Sp$n/testprompts
```

Create phone level master label files for the training utterances

```
>HLEd -l '*' -d dict -i Sp$n/phones0.mlf ../mkphones0.led
Sp$n/trainprompts.mlf
```

The remainder of the process can be performed for one speaker only. The other speakers in the group will be processed by a command file.

5. CODING THE SPEECH DATA

For one speaker Sp$n, create a list of waveform and mfcc files
`>../build_speech_mfcc_file_list Sp$n` (Ignore warning: rm: cannot remove 'Sp1/codetr.scp')

Code the data using HCopy
```
>HCopy -T 1 -C config.cfg -S Sp$n/codetr.scp
```

6. CREATING FLAT START MONOPHONES
Copy the prototype phone model proto.mmf from mother directory
```
>cp ../proto.mmf .
```

Compute global mean and variance of the prototype model
```
>HCompV -C config.cfg -f 0.01 -m -S Sp$n/train_data/files.scp
-M Sp$n/hmm0 ../proto.mmf
```

Copy the prototype model for each phone into Sp$n/hmm0/hmmdefs
```
>../proto2phones Sp$n
```

Split the model file into a macro and a clean model file
```
>echo "~o" >! Sp$n/hmm0/macros
>grep -i VECSIZE Sp$n/hmm0/hmmdefs >>! Sp$n/hmm0/macros
>cat Sp$n/hmm0/vFloors >>! Sp$n/hmm0/macros
```

Re-estimation to create initialised monophones
Create a list of the monophones in monophones except silence model 'sp'
```
>grep -v sp monophones1 >! monophones0
```

Three reestimation iterations. There will be warnings if some phones don't occur in the training data.
```
>HERest -C config.cfg -m 1 -I Sp$n/phones0.mlf -t 250.0 150.0
1000.0 -S Sp$n/train_data/files.scp -H Sp$n/hmm0/macros -H
Sp$n/hmm0/hmmdefs -M Sp$n/hmm1 monophones0
>HERest -C config.cfg -m 1 -I Sp$n/phones0.mlf -t 250.0 150.0
1000.0 -S Sp$n/train_data/files.scp -H Sp$n/hmm1/macros -H
Sp$n/hmm1/hmmdefs -M Sp$n/hmm2 monophones0
>HERest -C config.cfg -m 1 -I Sp$n/phones0.mlf -t 250.0 150.0
1000.0 -S Sp$n/train_data/files.scp -H Sp$n/hmm2/macros -H
Sp$n/hmm2/hmmdefs -M Sp$n/hmm3 monophones0
```

7. FIXING THE SILENCE MODELS
Copy the model and the macro files into hmm4/
```
>cp Sp$n/hmm3/hmmdefs Sp$n/hmm4
>cp Sp$n/hmm3/macros Sp$n/hmm4
```

Run the HTK-script "makesp" to create single-state silence model "sp" from "sil" and append it to the model file

```
>../makesp Sp$n/hmm3/hmmdefs >>! Sp$n/hmm4/hmmdefs
```

Set initial transition probabilities and tie the mid states of "sp" and "sil"

```
>HHEd -H Sp$n/hmm4/macros -H Sp$n/hmm4/hmmdefs -M Sp$n/hmm5
../sil.hed monophones1
```

Generate label files with short pauses between each word

```
>HLEd -l '*' -d dict -i Sp$n/phones1.mlf ../mkphones1.led
Sp$n/trainprompts.mlf
```

FURTHER RE-ESTIMATION

```
>HERest -C config.cfg -m 1 -I Sp$n/phones1.mlf -t 250.0 150.0
1000.0 -S Sp$n/train_data/files.scp -H Sp$n/hmm5/macros -H
Sp$n/hmm5/hmmdefs -M Sp$n/hmm6 monophones1
>HERest -C config.cfg -m 1 -I Sp$n/phones1.mlf -t 250.0 150.0
1000.0 -S Sp$n/train_data/files.scp -H Sp$n/hmm6/macros -H
Sp$n/hmm6/hmmdefs -M Sp$n/hmm7 monophones1
```

11. RECOGNITION AND EVALUATION

```
>HVite -C config.cfg -H Sp$n/hmm7/macros -H Sp$n/hmm7/hmmdefs
-S Sp$n/test_data/files.scp -l '*' -i Sp$n/hmm7/recout.mlf -w
wdnet -p 0.0 -s 5.0 dict monophones1
```

EVALUATION

```
>HResults -I Sp$n/testprompts.mlf monophones1
Sp$n/hmm7/recout.mlf
```

The above training and evaluation commands have been collected into a command script. This is provided by the teacher when the first speaker is processed.
It performs training and speaker-dependent evaluation for a speaker after each of the three reestimation iterations by

```
>cp ../train_rec.cmd .
>./train_rec.cmd Sp1
>./train_rec.cmd Sp2
```

Perform cross-speaker evaluation by recognising the test data from Sp1 using the training data from Sp2 and vice versa.

If you have time: Write a word loop grammar, repeat Sec 1., compute the perplexity and run the train_rec.cmd
Compare the recognition rates

If you have time: Running the recogniser live

```
HVite -C ../config2.cfg -H Sp$n/hmm7/macros -H
Sp$n/hmm7/hmmdefs -w wdnet -p 0.0 -s 5.0 dict monophones1
```

(Terribly slow)

POSSIBLE EXTENDED EXPERIMENTS

Change the number of cepstrum coefficients
The value of delta and acceleration parameters
Context-dependent phone models (triphones)
Multi-mixture models
Tying
Speaker independent recognition
Speaker Adaptation

Help

The above procedure is stored in the file Practical_commands.txt. Commands can be copied via cut-and-paste from this file into the command window.
Documents: HTK Book printout of the HTK Tutorial and the manual pages of the used commands.
Phoncode.doc: ARPAbet phoneme inventory.

Practical exercise report

Group number:

Group members
Sp1:
Sp2:
Sp3:

Grammar:

Perplexity:

Vocabulary
Vocabulary size:
Give phonetic transcription of words not in the BEEP dictionary:

Feature extraction conditions (defined in the configuration file)

Sampling frequency (kHz):	
Analysis window (ms):	
Frame interval (ms):	
Pre-emphasis coeff.	
Filterbank nbr channels	
Ceplifter	
Energy normalization	
Nbr cepstrum coefficients	
Hamming	

Acoustic model specification

Context-dependent phone models (yes/no)	
Tying (yes/no)	
Acoustic Feature type	
Acoustic vector size	
Nbr States per phone model	
Nbr mixture components per state	
Variance flooring factor (switch –f in HCompV)	

Recognition evaluation

Speaker dependent results: Matching the test speaker against his own trained models.
Cross-speaker results: Matching the test speaker against another speaker's models.

Possible extended tests:
Group dependent results: Average accuracy of each speaker matched against a model trained on all speakers in the group
All speakers group: Average accuracy of each speaker matched against a model trained on all speakers in all groups
Speaker independent results: Average of the accuracy for all test speakers. For each test speaker use his own grammar and an acoustic model trained on the training data of all the other speakers (round-robin, leave-one-out).

	Training speaker(s)	Test speaker(s)	Training size (#words)	Grammar perplexity	Test size (#words)	Accuracy (%)	#Ins.	#Del
Speaker dependent								
Speaker dependent								
Speaker dependent								
Cross-speaker								
Cross-speaker								
Cross-speaker								
Sp. Dep. Word loop								
(Group dependent)								
(All speakers group)								
(Ave. Speaker independent)								