Hidden Markov Model Toolkit (HTK) - Quick Start

by David Philippou-Hübner

This 5 page step-by-step tutorial shows how to build a (minimal) automated speech recognizer using HTK. It is a compact version of the HTK Tutorial Example (Chapter 3) but mainly concentrates on the single tools provided in HTK as well as on the essential (configuration) files. The tutorial is especially addressed to undergraduates who prefer a compact presentation of relevant steps in order to build a speech recognizing system. Hence, this tutorial raises no claim to completeness, but hopefully guaranties a quick lift.
For training merely one 7-word sentence (in German) is provided. The user should feel free to record and add further examples (using words from the provided lexica).
Hints refer to the additional scripts in order to use the HTK tools in a more convenient way. Further, they help to automatize recurring processes like the iterative training with HERest for instance. Note that for the java programs a Java-Runtime-Environment (JRE) is required.

0. Provide(d)
a) Training material, here: kino1.wav- kino5.wav ("Ich moechte heute Abend ins Kino gehen")
b) Pronunciation lexicon, here: german_lexicon.lex (for English: rm1_mono.lex)

1. Task Grammar
a) Create file „gram“ with following contents:
$word = ich | moechte | heute | Abend | ins | Kino | gehen;
(SENT-START <$word> SENT-END)

b) Create word network "wdnet" using HParse:
```
> HParse gram wdnet
```

2. Creating the dictinonary
a) Create a word list file "wlist" containing all words in the grammar in alphabetical order:
Abend
gehen
heute
ich
ins
Kino
moechte

b) Run HDMan:
```
> HDMan -m -w wlist -n monophones1 -l dlog dict german_lexicon.lex
```

Delivers the dictionary "dict", a list of occurring monophones "monophones1" and the log file "dlog" containing various statistics. Attention: the mapping between the words in "wlist" and pronunciation lexicon is case-sensitive, hence "Abend" and "abend" differ!

c) Edit "dict" by adding

SENT-END [] sil
SENT-START [] sil
silence sil

at the correct positions (so that the list remains sorted)

d) Check whether "monophones1" contains "sil" and "sp". If not, add

sil
sp

at the end of the file.

3. Preparing the recorded data

a) Create a word level MLF "words.mlf" containing the transcriptions of what is said in each .wav file and end each utterance with ".":

```
#!MLF!#
"*/kino1.lab"
ich
moechte
heute
Abend
ins
Kino
gehen
.
"*/kino2.lab"
ich
moechte
heute
Abend
ins
Kino
gehen
.
```

b) Create following edit script "mkphones0.led" containing:

```
EX
IS sil sil
DE sp
```

c) Generate the phone level MLF "phones0.mlf" via HLEd:

```
> HLEd -l '*' -d dict -i phones0.mlf mkphones0.led words.mlf
```

4. Coding the data

a) Create configuration file "config" with contents:

```
#Coding parameters
SOURCEKIND = WAVEFORMAT
SOURCEFORMAT = WAV
TARGETKIND = MFCC_0_D_A
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
```

WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F

b) Create "codetr.scp" script file to define for each .wav file the location and name of its corresponding .mfc file:

kino1.wav train/kino1.mfc
kino2.wav train/kino2.mfc
...

c) Make sure the path to the .mfc files exists (otherwise:> mkdir train) and run HCopy in order to extract the features:

```
> HCopy -T 1 -C config -S codetr.scp
```

5. Creating monophone HMMs
a) Create the "train.scp" file containing all files to be used in training:

train/kino1.mfc
train/kino2.mfc
...

b) Create a 3 state left to right HMM prototype called "proto", according to the HTK Tutorial page 30.

c) Disable the entries SOURCEKIND and SOURCEFORMAT in "config" by putting # in front of them!

d) Run HCompV:

```
> mkdir hmm0
> HCompV -C config -f 0.01 -m -S train.scp -M hmm0 proto
```

Hint:

```
> sh HCompV.sh hmm0 config train.scp proto
```

Delivers a new "proto" HMM in hmm0 and the file "vFloors".

e) Create "monophones0" file with the same content as "monophones1" except the "sp" entry.

f) Create "hmmdefs" file by copying the content of proto for each occurring monophone (in "monophones0"), such that each monophone has its own instance of a HMM in "hmmdefs". Modify the names of each HMM, e.g. ~h "ah" for the monophone "ah" HMM. Leave out all ~o entries as they will be set globally in the "macros" file.

Hint:

```
> java MakeHMMDefs proto monophones0
```

g) Create the "macros" file by opening "vFloors".

Add ~o <MFCC_0_D_A> <VecSize> 39 at the beginning and save it as "macros".

Hint:

```
> java GenerateMacros vFloors
```

h) Run HERest to re-estimate the HMM parameters:

```
> mkdir hmm1
> HERest -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp
      -H hmm0/macros -H hmm0/hmmdefs -M hmm1 monophones0
```

i) Repeat several iterations with HERest. Create a new hmm{step_number} directory for each run and be sure to adapt the directory-names in -H and -M options! Use the script below to easily repeat any number of iterations.

Hint:

```
> sh HERest.sh hmm0 {name_end_dir} {#iterations} config phones0.mlf
      train.scp monophones0
```

6. Fixing the silence models

a) Create the "sil.hed" script containing:

```
AT 2 4 0.2 {sil.transP}
AT 4 2 0.2 {sil.transP}
AT 1 3 0.3 {sp.transP}
TI silst {sil.state[3],sp.state[2]}
```

b) Open the latest "hmmdefs" file and copy it to a new hmm{latest+1} directory and add a one-state "sp" HMM. Make sure its topology is correct (<NumStates> 3, <TransP> 3, etc). The values of its only emitting state 2 are copied from the center state (state 3) of the "sil" HMM.

Hint:

```
> java Add_SP  hmmdefs {name_of_destination_file}
```

c) Run HHEd:

```
> mkdir hmm{latest+1}
> HHEd -H hmm{latest}/macros -H hmm{latest}/hmmdefs
      -M hmm{latest+1} sil.hed monophones1
```

d) Repeat several iterations with HERest as in 5i), but using "monophones1" as last parameter.

7. Realigning the data [optional]

This is for re-estimation in order to include different pronunciation variants.

a) Run HVite:

```
> HVite -l '*' -o SWT -b silence -C config -a -H hmm{latest}/macros -H
      hmm{latest}/hmmdefs -i aligned.mlf -m -t 250.0 -y lab -I words.mlf
      -S train.scp dict monophones1
```

b) Repeat several iterations with HERest as in 6d).

8. Recognizer Evaluation

a) Create the "test.scp" file containing all files to be used in the test (for a first trial this file may be identical with "train.scp").

train/kino1.mfc
train/kino2.mfc
...

b) Run HVite and HResults. The final recognition results are saved in recout.mlf:

```
> HVite -H hmm{latest}/macros -H hmm{latest}/hmmdefs -S test.scp -l '*'
        -i recout.mlf -w wdnet -p 0.0 -s 5.0 dict monophones1

> HResults -I words.mlf monophones1 recout.mlf
```

Hint:

```
> sh HVite.sh hmm{latest} test.scp recout.mlf wdnet dict monophones1
        words.mlf
```

I hoped you enjoyed this tutorial and that it was a helpful starting point in order to proceed with your own research.

David Philippou-Hübner
Chair of Cognitive Systems; Institute for Information and Communication Technology; Faculty of Electrical Engineering and Information Technology
Otto von Guericke University Magdeburg
Universitätsplatz 2, D-39106 Magdeburg, Germany