ICFHR–2014 TUTORIAL

"Handwritten Text Recognition: Word-Graphs, Keyword Spotting and Computer Assisted Transcription"

Practice Session

Moisés Pastor, Joan Andreu Sánchez, Alejandro H. Toselli and Enrique Vidal

Pattern Recognition & Human Language Technology Research Center
Universidad Politécnica de Valencia
EU 7th FP tranScriptorium project (Ref: 600707)
[mpastorg,jandreu,ahector,evidal]@prhlt.upv.es


Demokritos, September 1st, 2014

1 Introduction

The aim of this practice guide is that the students get familiar with the use of **HTK** (Hidden Markov Model ToolKit) applied in handwritten text recognition (HTR). Likewise, it will be show how to obtain word-graphs from the HTR decoding process and how them can be used for parameter optimization or for decoding using n-gram models with $n > 2$ (re-scoring). In addition, brief explanations about the use of some homemade tools for image preprocessing and features extraction implemented for HTR will be given.

As this practice is completely developed in Linux, it is assumed that the students have a prior knowledge and experience using this operating system and handling the standard GNU-Linux tools such as: `bash, awk, sed, etc`. For more information of these tools, refer to:

- `man bash awk sed`
- http://www.gnu.org/software/bash/manual/bashref.html
- http://www.faqs.org/docs/Linux-HOWTO/Bash-Prog-Intro-HOWTO.html
- http://www.gnu.org/software/gawk/manual/gawk.html
- https://www.gnu.org/software/sed/manual/sed.html

2 HTK and Software Tools for HTR

The **HTK** tool, as well as its documentation, are publically available in:

- http://htk.eng.cam.ac.uk

In addition, in order to train n-grams language model, the software *SRI Language Modeling Toolkit* (SRILM) is required This can be downloaded from:

- http://www.speech.sri.com/projects/srilm/download.html

Create a work directory for this practice session and enter it. For example:

```
mkdir $HOME/work
cd $HOME/work
```

The *tools*, *utils* and *examples* for HTR can be downloaded from:

```
wget http://transcriptorium.eu/~tutorialICFHR/\
     DOCUMENTATION/HTR-toolsUtils.tar.bz2 \
     --http-user=icfhr2014 --http-passwd=icfhr2014
```

It is needed to have installed the image libraries `libjpeg, libtiff and libpng`. To compile and install it:

```
mkdir bin
tar xvjf HTR-toolsUtils.tar.bz2
cd HTR-toolsUtils/src
./compile.sh ../../bin
cd ../../
```

In "`$HOME/work/bin`" are found the following executable files:

`imageSlope:` detects and corrects the angle of the handwritten text line with respect to the horizontal direction by a rotation transform.

`imageSlant:` detects the words slants in the image and corrects them by an horizontal shift transform (see [2]).

`pgmnormsize:` performs size normalization of each detected word in the image (see [3]).

`featExtraction:` transforms the already preprocessed image into sequence of features vectors (see [1]).

In "`$HOME/work/HTR-toolsUtils/scripts`" has been installed the following bash-shell scripts:

`HTR-prep-feat.sh` : launches the complete text image preprocessing and features extraction on a given corpus of text lines images.

`HTRfeatShow.sh` : build a graphical representation for a given feature file (with extension `.fea`). This script places an image file in the current directory path (with extension `_fea.pgm`), containing a visual representation of the three types of features: gray-level and, horizontal and vertical derivatives.

`Create_HMMs-TOPOLOGY.sh` : sets the initial HMM topology.

`Create_HTK-DicNet.sh` : Given a directory containing the transcriptions files, this script generates a `Dictionary` and a `Network` (i.e. language model) in the SLF format of **HTK**. This script runs the *ngram-count* binary (provided by the SRILM ToolKit) to generate the language model (bi-grams).

`Create_ListOfHMMs.sh` : generates a characters list, (which serves as HMMs identifiers) from a given directory of transcriptions files.

`Create_Train-MLF.sh` : accepts as input a directory of transcriptions files (in text format) and outputs a **HTK** MLF (Master Label File), which is going to be used in the HMMs training process.

`Create_Test-MLF.sh` : accepts as input a directory of transcriptions files (in text format) and outputs a **HTK** MLF (Master Label File), which is going to be used as reference labels in the final evaluation of the recognized hypotheses.

`Train-HMMs.sh` : launches a complete HMMs training process. For a more detailed explanation of what this script does, refer to the appendix A.

In order to these tools (executables and scripts) are available to the user-command line, do not forget to include their respective paths into the shell `PATH` variable:

```
WORK=$HOME/work
HTK_BIN=... (absolute path to the HTK binaries)
SRILM_BIN=... (absolute path to the SRILM binaries)
export PATH=$PATH:$WORK/bin:$WORK/HTR-toolsUtils/scripts:\
           $HTK_BIN:SRILM_BIN:.
```

You can have a look into each of these scripts to learn more about what exactly they are doing. Mostly of them employ **HTK** commands, although the `Create_HTK-DicNet.sh` script, also uses some **SRILM** commands to train n-grams models.

3 Corpus "Bentham Collection"

This task is composed by a set of manuscripts written by the English philosopher and reformer Jeremy Bentham[1] and some copies written by his secretarial staff. The manuscripts covers different subjects as legal reform, punishment, the constitution, religion, and his panopticon prison scheme. The Bentham collection was written without any standard and because that they have some difficulties. Belong them it can be found deletions, marginalia, interlineal additions, crossed out words, etc.
The dataset used in this tutorial is a subcorpus of the one that was used in the ICFHR 2014 Handwritten Text Recognition in the `tranScriptorium Dataset` (HTRtS) contest[2] (freely available for research purposes from the tranScriptorium web page http://transcriptorium.eu).
The subcorpus can be downloaded from:

```
wget http://transcriptorium.eu/~tutorialICFHR/\
          DOCUMENTATION/BenthamData.tar.bz2 \
          --http-user=icfhr2014 --http-passwd=icfhr2014
```

and uncompressed using:

```
tar xvjf BenthamData.tar.bz2
```

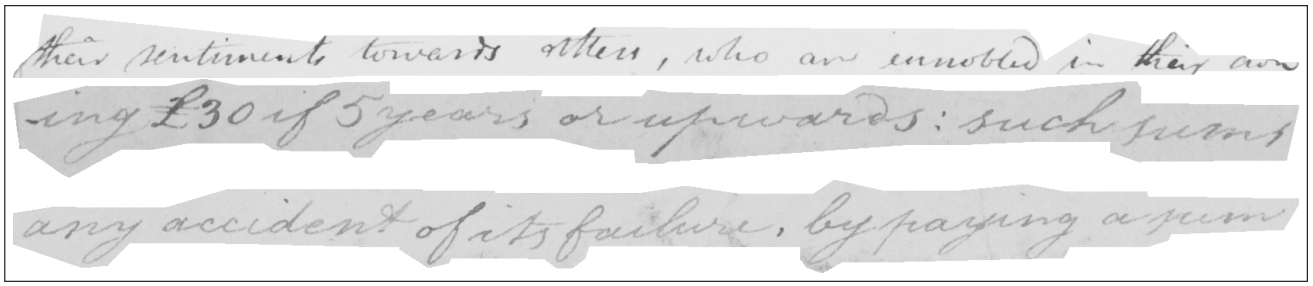

Figure 1: Some text lines examples from the Bentham corpus.

Into the directory provided for this tutorial, `BenthamData` you can find:

1. The sub-directory "`Line-Images`", which contains the line images in PNG format. Figure 2 shows some examples of them.

2. The sub-directory "`GT-Transcrips`", where the corresponding ground truth transcriptions files (in ASCII format) are located.

3. The sub-directory "`Exp-Stuff`", where are placed the files with the lists of the training ("`train.lst`") and test ("`test.lst`") images. Both of them list the samples without directories neither extensions. Moreover, in this directory it is also the file "`large-text.dat`" used for training 3-gram language model for re-scoring word-graphs (see Sec.5.2).

   The table 1 shows basic statistics of the corpus as well as the defined partition.

[1] http://blogs.ucl.ac.uk/transcribe-bentham/jeremy-bentham
[2] http://transcriptorium.eu/~htrcontest

	Train	Test	Total
# sentences	1 193	150	1 343
# words	10 519	1 441	11 960
# chars	54 276	7 648	61 924

Table 1: Basic statistics from the corpus and its defined partition.

4 Tutorial for a Simple HTR experiment

The HTR experiment described here, involves five different phases: **Preprocessing and Feature extraction**, **HMM training**, **building language model**, **decoding** and **evaluation**. Furthermore, an introduction of how to use **Word-graphs** for tuning fast and easily the `Grammar Scale Factor` and `Word Insertion Penalty`, together with an example of how to use WGs for re-scoring using trigrams.

4.1 Preprocessing and Feature extraction

The steps to be followed are:

1. `mkdir Features`
2. Apply the HTR preprocessing and features extraction on the "Bentham" images located in "`BenthamData/Line-Images`". To do this, we use the script:

   ```
   HTR-prep-feat.sh BenthamData/Line-Images Features
   ```

   The `HTR-prep-feat.sh` takes as arguments:

   (a) the directory containing the text line images ("`BenthamData/Line-Images`") .

   (b) the output directory ("`Features`") where the features files will be stored.

   This script performs for each of the images, the following sequence of piped commands:

   ```
   imgtxtenh -i $f -S 4 -a|  # clean and enhance the image.
   imageSlope |              # Perform "Slope" correction.
   imageSlant -m M -t 92|    # Perform "Slant" correction
                             #   using Standard Deviation.
   pgmnormsize -c 5|         # Perform size normalization.
   featExtraction -c 20 \
       -V 2 -O 2 -F 1
       -H -o name.fea"
                             # Compute feature extraction.
   ```

   Furthermore, to check if **HTK** accepts this feature extraction format-type, list the content of this file through the command **HList**, as follows:

   ```
   HList -h Features/071_003_004_04_03.fea
   ```

To test whether the feature extraction files are being correctly generated, we can display graphically one of them through the script "`HTRfeatShow.sh`". For example:

```
HTRfeatShow.sh Features/073_071_001_03_13.fea
```

This script should create an image, "`./073_071_001_03_13_fea.pgm`", with a visual representation of the gray-level feature and both the horizontal and vertical derivative features[3].

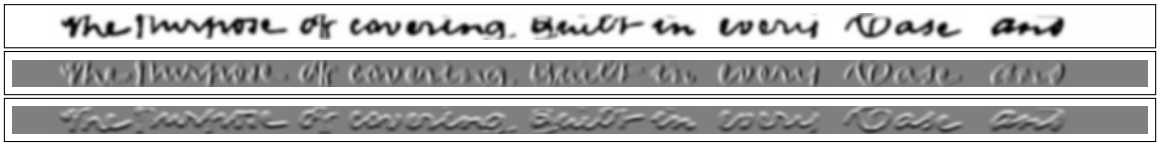

Figure 2: Some text lines examples from the Bentham corpus.

The execution of the `HTR-prep-feat.sh` script for all the images takes around 20 minutes to finish, so in the mean time we can go on with the next step.

4.2 HMMs Training

The steps to be followed in this training process are:

1. Run the script `Create_ListOfHMMs.sh` to generate the HMMs names list along with the defined number of HMM states:

   ```
   Create_ListOfHMMs.sh BenthamData/GT-Transcrips 8 HMMsList
   ```

   As arguments, the directory containing the transcriptions files, the number of states and the name of the output file (`HMMsList`) are required.

   To see the `HMMsList` content,

   ```
   cat HMMsList
   ```

   It must be noted that for some HMMs as the punctuations marks, the number of states is reduced to half. Feel free to edit `HMMsList` and decide the number of stated you consider.

2. The HMM training process applied here, requires (among others things) the feature extraction files and the corresponding transcriptions. Concerning to the transcriptions file, it must be in MLF (master label format) and can obtained by executing the `Create_Train-MLF.sh` script on them:

   ```
   Create_Train-MLF.sh BenthamData/GT-Transcrips samples.mlf
   ```

   The output is stored in the "`samples.mlf`" file.

3. From the "`BenthamData/Exp-Stuff/training.lst`" file, we generate the final training samples list "`train.lst`" as follows:

[3] In this image the edges have been emphasized.

```
for m in $(< BenthamData/Exp-Stuff/training.lst); do
  if [ -e Features/$m.fea ]; then
    echo Features/$m.fea;
  else
    echo "Error: Features/$m.fea does not exist." 1>&2;
  fi;
done > train.lst
```

4. Finally (once the processing and feature extraction have finished for all images), we run the following script to train the HMMs:

```
Train-HMMs.sh train.lst hmms samples.mlf HMMsList 4 16 60
```

This script accepts as arguments:

(a) the list of samples files to be used in the training process.

(b) the directory where the trained HMMs will be stored.

(c) the transcriptions file in MLF format.

(d) the file of HMMs names list along with its defined number of states.

(e) number of iterations.

(f) final number of Gaussian densities per HMM state.

(g) dimension of the features vectors that (in this case) is set to $60 = 20 \times 3$, according to the the feature extraction resolution of 20 computed in the preprocessing step (20 grey level features, 20 horizontal derivative features and 20 vertical derivative features).

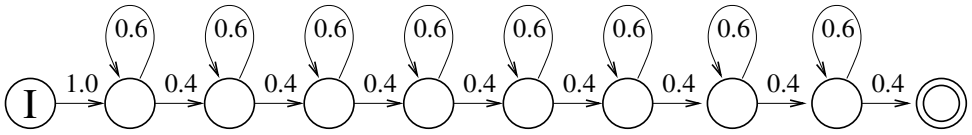

Figure 3: Example of HMM, 8 states left-to-right topology.

The HMMs topology has the state-transition to itself with probability 0.6 and to the next state (on the right side) with probability 0.4 (see an example on Fig. 3).

During the execution of the script "`Train-HMMs.sh`", it is created the "`AuxHMMsList`" file from the "`HMMsList`" containing only the first column, that is, the HMM names.

For a more detailed explanation of what this script exactly does, refer to the appendix A.

The execution of the `Train-HMMs.sh` script takes around 40 minutes to finish, so a HMMs model set trained is provided.

4.3 Building Language Model and Dictionary

To restrict the HTR outputs to a set of valid words the decoding process employs a dictionary. Likewise, the use of language model helps in restricting the search space to grammatically well-formed and meaningful sentences. In this case, we employ an n-gram model which is a type of probabilistic language model estimated from a large text corpus which is related to the task at hand.

To build the dictionary and the n-gram model from the training ground-truth transcripts, we proceed to run the `Create_HTK-DicNet.sh` script: Create the words dictionary and the network (language model) using:

```
Create_HTK-DicNet.sh BenthamData/Exp-Stuff/training.lst \
                     BenthamData/GT-Transcrips \
                     Dictionary Network
```

The first argument is the "`training.lst`" file listing the 1 193 training samples while the second argument specified the directory path containing its transcription. The third and fourth arguments are the names of the dictionary and the network output files respectively. The network file (`Network`) is in SLF (standard lattice format) of **HTK**. Actually this script runs the binary `n-gram-count` (belonging to the SRILM Toolkit) to train a bi-grams language model "`LM.arpa`" (in ARPA format) from the training file "`train_samples`" compiled from the training ground-truth transcripts.

```
ngram-count -text train_sentences -lm LM.arpa -order 2 \
            -ukndiscount1 -ukndiscount2
```

One way to test whether these files have been correctly generated, is employing the command **HSGen**, which generates randomly several sentences using the lexicon and the language model probabilities from the "`Dictionary`" and the "`Network`" files.

```
HSGen -n 100 Network Dictionary
```

4.4 Recognition Phase

1. In a similar way, as it was done for the "`train.lst`" file, we create the test samples list "`test.lst`":

```
for m in $(< BenthamData/Exp-Stuff/test.lst); do
  if [ -e Features/$m.fea ]; then
    echo Features/$m.fea;
  else
    echo "Error: Features/$m.fea does not exist." 1>&2;
  fi;
done > test.lst
```

2. Finally, we perform the proper recognition through the **HTK** command **HVite**. For reasons of time, we only perform the recognition of two samples ("`Features/097_186_001_01_23.fea''` and "`Features/115_077_003_02_20.fea''`) to show how it is carried out. However, the way of executing **HVite** for the whole list of samples (`test.lst`) is given in the appendix B.

```
mkdir WGs-2grms
HVite -A -T 1 -o ST -p -40 -s 40 -n 15 1 -z lat -q Atal \
      -H hmms/hmm_16/Macros_hmm -l WGs-2grms -i res16.mlf \
      -w Network Dictionary AuxHMMsList \
       Features/097_186_001_01_23.fea \
       Features/115_077_003_02_20.fea
```

In this case, the recognition is carried out using the trained HMMs with 16 Gaussians per state defined in "hmms/hmm_16/Macros_hmm" file and the lexicon and language model defined respectively in the "Dictionary" and "Network" files. As commented in the HMM training sec.4.2, "AuxHMMsList" was generated during the execution of the the script "Train-HMMs.sh". The arguments "-s 40" and "-p -40" set up the *grammar scale factor* and *word insertion penalty* for the recognition process. For more information about the remaining arguments loop up the **HTK** manual. The recognized hypotheses are stored in the "res16.mlf" file while their corresponding word-graphs in the "WGs-2grms" dir: "097_186_001_01_23.lat" and "115_077_003_02_20.lat". To see them and their respective references:

```
cat res16.mlf
cat BenthamData/GT-Transcrips/{097_186_001_01_23.txt,\
                               115_077_003_02_20.txt}
```

As we observe, these results are not so good, since they were recognized using HMMs with not enough number of Gaussians per mixture and trained with few samples.

4.5 Evaluation Phase

Once the recognition has finished, we can assess the accuracy of the recognized hypotheses. To do this the two following steps have to be carried out:

1. Create the reference transcriptions file (in MLF format) "SamplesRef.mlf":

```
Create_Test-MLF.sh BenthamData/GT-Transcrips SamplesRef.mlf
```

2. Get the scores by using the command **HResults**:

```
HResults -t -I SamplesRef.mlf AuxHMMsList res16.mlf
```

which outputs:

```
Aligned transcription: 097_186_001_01_23.lab vs 097_186_001_01_23.rec
 LAB: will be a matter of no immediate concern to me ; if I succeed only
 REC: will be a matter of no immediate not     to me ; if I succeed only
====================== HTK Results Analysis =======================
  Date: Tue Jul 15 14:15:46 2014
  Ref : SamplesRef.mlf
  Rec : res16.mlf
------------------------ Overall Results --------------------------
SENT: %Correct=50.00 [H=1, S=1, N=2]
WORD: %Corr=95.83, Acc=95.83 [H=23, D=0, S=1, I=0, N=24]
===================================================================
```

where Acc=95.83 is the accuracy rate corresponding to WER=4.17% (word error rate). The WER is defined by:

$$\text{WER} = 100 - \text{Acc} = \left(\frac{D+S+I}{N}\right) \cdot 100$$

where:

I : number of insertions
D : number of deletions
S : number of substitutions
H : number of correct labels
N : total number of word references in the defining transcription files.

5 Word Graphs

5.1 Tuning Grammar Scale Factor and Word Insertion Penalty

This section presents a fast WG-based approach for optimizing two parameters related with the recognition process: *Grammar Scale Factor* (GSF) and *Word Insertion Penelty* (WIP).

1. Build the list of previous WGs generated using 2-gram language model:

```
ls WGs-2grms/*.lat > listLats.lst
```

2. Create a config file named "`config_HLRescore`" required by the `HLRescore` command, which has to specify the START and END word sentence symbols (`<s>` and `</s>`):

```
echo -e "STARTWORD = <s>\nENDWORD = </s>" > config_HLRescore
```

3. To obtain new recognized hypotheses using the 2-gram WGs by specifying different values of GSF and WIP (as 5.8 and 6.0 respectively according the below example), we run the following command:

```
HLRescore -C config_HLRescore -o ST -s 5.8 -p 6.0 -f \
          -i tmp.mlf -S listLats.lst Dictionary
```

   The parameter values of `-s` and `-p` must be "floats".

4. As before, we can evaluate the new decoding performance by running:

```
sed -r -e "/<s>|<\/s>/d" \
 -e "s/^[0-9]+$/\"&\"/" tmp.mlf > new_res.mlf
HResults -t -I SamplesRef.mlf AuxHMMsList new_res.mlf
```

5. Here is an example of shell script of how to tune GSF and WIP using previous generated WGs:

```
for s in 20.0 30.0 40.0 50.0 60.0; do
  for p in -20.0 -30.0 -40.0 -50.0 -60.0; do
    echo -e "\ns="$s " p="$p
    HLRescore -C config_HLRescore -o ST -s $s -p $p -f \
              -i tmp.mlf -S listLats.lst Dictionary
    sed -r -e "/<s>|<\/s>/d" \
           -e "s/^[0-9]+$/\"&\"/" tmp.mlf > new_res.mlf
    HResults -I SamplesRef.mlf AuxHMMsList new_res.mlf
  done
done
```

5.2 Re-scoring using Trigrams

1. Generate the "`3grams.arpa`" language model using the collected large plain ASCII text of Bentham transcriptions (excluding the ones belonging to the test partition), placed in `BenthamData/Exp-Stuff/large-text.dat`:

```
cat BenthamData/Exp-Stuff/large-text.dat train_sentences |
ngram-count -text - -lm - -order 3 -ukndiscount |
sed "s/['\"]/\\\&/g" > 3grams.arpa
```

2. Create a new directory for placing the re-scored WGs:

```
mkdir WGs-3grms
```

   which are obtained using again the `HLRescore` command by applying the 3-gram LM "`3grams.arpa`" on the previous WGs stored in "`WGs-2grams`" directory:

```
HLRescore -T 1 -A -C config_HLRescore -s 40.0 -p -40.0 \
          -S listLats.lst -n 3grams.arpa -w \
          -l WGs-3grms -q Atal Dictionary
```

3. Similar as was done in the previous section, the list of 3-gram WGs is obtained by:

```
ls WGs-3grms/*.lat > listLats-3grams.lst
```

4. And for getting the best hypotheses from the 3-gram WGs:

```
HLRescore -A -T 1 -C config_HLRescore -o ST \
          -s 40.0 -p -40.0 -f -i tmp.mlf \
          -S listLats-3grams.lst Dictionary
```

5. Finally, new performence evaluation result (WER):

```
  sed -r -e "/<s>|<\/s>/d" -e "s/^[0-9]+$/\"&\"/" \
         tmp.mlf > rec_3grams.mlf
  HResults -I SamplesRef.mlf AuxHMMsList rec_3grams.mlf
```

References

[1] I. Bazzi, R. Schwartz, and J. Makhoul. An Omnifont Open-Vocabulary OCR System for English and Arabic. *IEEE Trans. on PAMI*, 21(6):495–504, 1999.

[2] Moisés Pastor, Alejandro Toselli, and Enrique Vidal. Projection Profile Based Algorithm for Slant Removal. In *International Conference on Image Analysis and Recognition (ICIAR'04)*, Lecture Notes in Computer Science, pages 183–190, Porto, Portugal, September 2004. Springer-Verlag.

[3] V. Romero, M. Pastor, A. H. Toselli, and E. Vidal. Criteria for handwritten off-line text size normalization. In *Procc. of The Sixth IASTED international Conference on Visualization, Imaging, and Image Processing (VIIP 06)*, Palma de Mallorca, Spain, August 2006.

[4] S. Young, J. Odell, D. Ollason, V. Valtchev, and P. Woodland. *The HTK Book: Hidden Markov Models Toolkit V2.1*. Cambridge Research Laboratory Ltd, March 1997.

Appendices

A HMMs Training

In the HMMs training process, the following steps take place:

1. HMMs Initialization:

```
mkdir -p hmm/hmm_0
HCompV -A -T 1 -f 0.01 -m \
       -S train_red.lst -M hmm/hmm_0 proto
```

   The **HCompV** command computes the global mean and variance of the whole training samples set, and puts them into "hmm/hmm_0/proto" file. In addition, the `-f 0.01` argument creates the "hmm/hmm_0/vFloors" file containing a floor variance computed as a percent (0.01) of the global variance. In the HMMs training process, because of lacking enough training samples, there could be cases of Gaussians whose variance components fall down below a previously set threshold. In such cases, these components would be substituted by the respective floor variance components.

   Once the global mean and variance have been computed and written into the "hmm/hmm_0/proto" file, next step is the generation of the *master macro file* (MMF) containing all the HMMs, listed in the file "HMMsList", with their respective parameters initialized with the global mean and variance values. This is achieved in the following way:

```
mkdir -p hmm/hmm_1
head -3 hmm/hmm_0/proto > hmm/hmm_1/Macros_hmm
cat hmm/hmm_0/vFloors >> hmm/hmm_1/Macros_hmm
for i in $(< HMMsList); do
  tail -n +4 hmm/hmm_0/proto |
  sed "s/proto/$i/g" >> hmm/hmm_1/Macros_hmm
done
```

2. HMMs training with a mixture of one Gaussian per state is carried out by:

```
k=1; while [ $k -le 4 ]; do
  HERest -A -T 1 -m 3 -S train.lst \
         -I sample.mlf -H hmm/hmm_1/Macros_hmm HMMsList
  k=$[k+1]
done
```

   As can be seen, four iterations are made, where the HMMs parameters values are read, re-estimated and written again to hmm/hmm_1/Macros_hmm.

3. Once the HMMs (defined in hmm/hmm_1/Macros_hmm) with one Gaussian in the mixture per state have been trained, it is proceeded to duplicate the number of Gaussians for these all HMMs:

```
mkdir -p hmm/hmm_2
echo "MU 2 {*.state[2-7].mix}" > mult_script
HHEd -A -H hmm/hmm_1/Macros_hmm -M hmm/hmm_2 \
                        mult_script HMMsList
```

The command **HHEd** is in charge to do this task. It takes as input a script file containing the line "MU X {*.state[2-D].mix}", where D is the number of the HMMs emission states plus one (8+1 in this case) plus 1, and X would be equal to 2 if we want to duplicate the number of Gaussian, to 4 if we want to quadruplicate them, and so on. For more detailed information about the syntax of this command, see the **HTK** manual [4]. For the later example, the new MMF with the duplicate Gaussians, will be stored in `hmm/hmm_2/Macros_hmm`.

The last two steps (2 and 3) are repeated iteratively obtaining in this way MMF with HMMs trained for different number of Gaussians in the mixture of each state.

B Performing a Complete Recognition and WG Generation Process given a List of Features Files

Instead of appending the names of files to recognized (at the end of the **HVite** command), we use the option: "`-S test.lst`" as follows:

```
HVite -A -T 1 -o ST -p -40 -s 40 -n 15 1 -z lat -q Atal \
      -S test.lst -H hmms/hmm_16/Macros_hmm -l WGs-2grms \
      -i res16.mlf -w Network Dictionary AuxHMMsList
```

In this case the recognition is carried out on the samples files listed in `test.lst` file, using the trained HMMs with 16 Gaussians per state (defined in `hmms/hmm_16/Macros_hmm` file) and the lexicon and language model defined respectively in the "`Dictionary`" and "`Network`" files. Each recognized hypothesis is stored in the "`res16.mlf`" file. The arguments "`-s 40`" and "`-p -40`" set up the grammar scale factor and word insertion penalty for the recognition process. The recognized first best hypoteses are written to the file `res16.mlf`, while the corresponding word-graphs are placed in the `WGs-2grms` directory. For more information about the remaining arguments loop up the **HTK** manual.