

ICFHR–2010 TUTORIAL

“Multimodal Computer Assisted Transcription of
of Handwriting Images”

Practice Session

Alejandro H. Toselli, Moisés Pastor and Verónica Romero

Instituto Tecnológico de Informática

Universidad Politécnica de Valencia

[ahector,moises,vromero]@iti.upv.es



November 15th, 2010

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), and further, in computer assisted transcription of handwritten text (CATTI). In addition, brief explanations about the use of some home-made 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`, `netpbm`, `display`, etc. For more information of these tools, refer to:

- `man bash awk netpbm display`
- <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>
- <http://netpbm.sourceforge.net/>
- <http://netpbm.sourceforge.net/doc/>
- <http://www.imagemagick.org/script/display.php>

These tools are employed to perform basic tests and HTR experiments on the small handwritten text images corpus named *Spanish-Numbers*. The recognition of text images from this corpus corresponds to an academic problem with application in the real field, focussing on the recognition of legal amounts in bank checks. That is, these handwritten text sentences simulate legal amount numbers in bank checks (see examples in Fig. 1).

2 HTK and Software Tools for HTR and CATTI

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`

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

`wget http://prhlt.iti.es/tutorial/icfhr2010/HTR-toolsUtils.tar.bz2`

To compile and install it:

```
tar xvjf HTR-toolsUtils.tar.bz2
cd HTR-toolsUtils
make
make install
```

By default, they will be installed in “`$HOME/bin`”, “`$HOME/scripts`” and “`$HOME/CATTI-Examples`”. In “`$HOME/bin`” are found the following executable files:

pgmmmedian: performs median filter for noise reduction and edge smoothing.

pgmslope: by a rotation transform.

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

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

pgmtextfea: transforms the already preprocessed image into sequence of features vectors.

grammarTools: is in charge to generate the n -grams language model required for each CATTI interaction. Actually, this program performs the “concatenation” of a *linear* model which strictly accounts for the successive words in the prefix and the n -grams models which models the words sequence in the suffix [?].

pfl2htk: converts sequence of real-values vectors in ASCII format to HTK format.

In “\$HOME/scripts” has been installed the following bash-shell scripts:

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

HTRfeatShow.sh : shows a graphical representation of a given features extraction file (with extension .fea). To run this script is necessary to have `display` previously installed (all Linux have it installed by default).

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 SLF format of **HTK**. This script runs the *ngram-count* binary (provided by the SRILM Toolkit) to generate the language model (bi-grams).

Create_HMMsList.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.

CATTI_simulation.sh : launches a CATTI simulation for a given set of samples. This will be explained in section 5.

shuffle.sh : shuffles the lines of a given file.

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:

```
export PATH=$PATH:$HOME/bin:$HOME/scripts:.
```

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.

Finally, the “\$HOME/CATTI-Examples” directory has the “CATTI_IAMDB.log” file, which contains the CATTI simulation output on the IAMDB corpus. This file is going to be used in the proposed exercises.

3 Corpus “Spanish-Numbers”

The “Spanish-Numbers” corpus [?], collected by the “Instituto Tecnológico de Informática¹”, contains about 522 handwritten text sentences and is employed frequently as example-task for assessing the performance of new preprocessing, features extraction and modelling methods for HTR. It can be downloaded from:

```
cd $HOME
wget http://prhlt.iti.es/tutorial/icfhr2010/SpanishNumbers.tar.bz2
```

and uncompressed using:

```
tar xvjf SpanishNumbers.tar.bz2
cd $HOME/SpanishNumbers
```

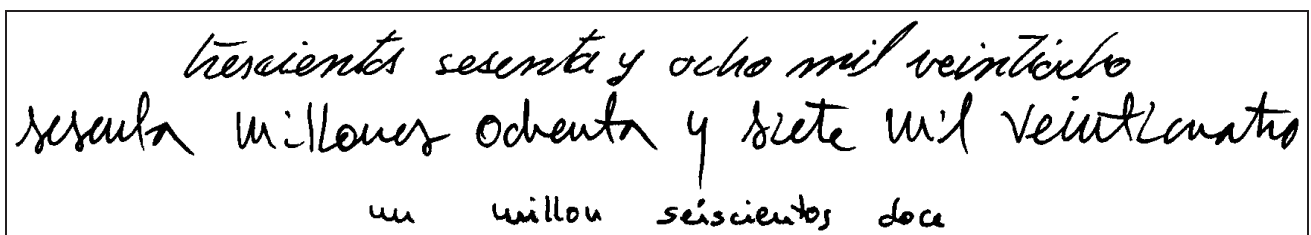


Figure 1: Some text lines examples from the “Spanish-Numbers” corpus.

Into the directory `SpanishNumbers` you can find:

1. The sub-directory “FRASES_PBM”, which contains the 522 binary images in PBM format. Figure 1 shows some examples of them.
2. The sub-directory “FRASES_TXT”, where the corresponding transcriptions files (in ASCII format) are located.
3. The sub-directory “INF”, where are placed the files with the samples lists used for training (“`list.train`”) and test (“`list.train`”), as well as the “`samplesDB.dat`” file used to generate the dictionary (lexicon) and the language model for the recognition.

The files “`list.train`” and “`list.test`” define the training and test partitions respectively. Both of them list the samples without directories neither extensions, located in “`SpanishNumbers/FRASES_PBM`” and “`SpanishNumbers/FRASES_TXT`”. The table 1 shows basic statistics of the corpus as well as the defined partition.

	Train	Test	Total
# writers	18	11	29
# sentences	297	187	484
# words	1,300	827	2,127

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

¹<http://www.iti.upv.es>

4 Tutorial for a Simple HTR experiment

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

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

The HTR experiment described here, involves three different phases: **training**, **decoding** and **evaluation**.

4.1 Training Phase

The steps to be followed in this training process are:

1. `mkdir $HOME/work/feaDir`
2. Apply the HTR preprocessing and features extraction on the “Spanish-Numbers” images located in “`SpanishNumbers/FRASES_PBM`”. To do this, we use the script: `feaDir 20`

```
HTR-Preprocessing.sh ../SpanishNumbers/FRASES_PBM feaDir 20
```

The `HTR-Preprocessing.sh` takes as arguments:

- (a) the directory containing the images in PBM format.
- (b) the output directory (“`feaDir`”) where the features extraction files will be stored.
- (c) the features extraction resolution (20).

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

```
pnmdepth 255 file.pbm |           # Change to grey Level.
pgmmedian |                   # Smooth the image.
pgmslope |                     # Perform "Slope" correction.
pgmslant -M M |                # Perform "Slant" correction
                               # using Standard Deviation.
pgmnormsize -c 5 |             # Perform size normalization.
pgmtextfea -c $RES > file.aux # Compute feature extraction.
pfl2htk file.aux file.fea      # Convert to HTK format.
```

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 feaDir/010002.fea
```

This script should display three images of the text image “*un millon setecientos mil nueve*” (1,700,009). The first should correspond with the normalized grey level features, whereas the second and third with the horizontal and vertical derivatives features respectively².

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

²In this images the edges have been emphasized.

```
HList -h feaDir/010002.fea
```

The execution of the `HTR-Preprocessing.sh` script for all the images takes around 15 minutes to finish, so in the mean time we can go on with the next step.

3. The next step is to generate a HMM prototype with an initial continuous density *left-to-right* topology using:

```
Create_HMMs-TOPOLOGY.sh 60 6 > proto
```

This script accepts as input the dimension of the features vectors and the number of states to be set up for HMMs. It outputs an initial HMM topology with only one Gaussian density in the mixture per state, whose dimension (in this case) is set to $60 = 20 \times 3$, according to the feature extraction resolution of 20 computed in the preprocessing step (20 grey level features, 20 horizontal derivative features and 20 vertical derivative features). The number of states per HMM has been set up to 8, each of them has the state-transition to itself with probability 0.6 and to the next state (on the right side) with probability 0.4.

To take a look into the initial HMM prototype `proto`,

```
cat proto
```

4. Run the script `Create_HMMsList.sh` to generate the HMMs names list:

```
Create_HMMsList.sh ../SpanishNumbers/FRASES_TXT \  
HMMsList
```

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

```
cat HMMsList
```

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

```
Create_Train-MLF.sh ../SpanishNumbers/FRASES_TXT \  
samples.mlf
```

The output is stored in the “`samples.mlf`” file.

6. From the “`SpanishNumbers/INF/list.train`” file, we generate the final training samples list “`train.lst`” as follows:

```
for m in $(< ../SpanishNumbers/INF/list.train); do  
  if [ -e feaDir/$m.fea ]; then  
    echo feaDir/$m.fea;  
  else  
    echo "Error: feaDir/$m.fea does not exist." 1>&2;  
  fi;  
done > train.lst  
shuffle.sh train.lst | tail -n ${297/4} > train_red.lst
```

The last line of the previous script performs a random selection of a sub-set of the training samples list “train.lst”. We will employ this reduced list (train_red.lst) to speed up the training process, at the expense of the HMMs being not well trained.

7. 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_red.lst hmm proto samples.mlf \  
HMMsList
```

This accepts as arguments:

- (a) the list of samples files to use in the training process.
- (b) the directory where will be stored the trained HMMs.
- (c) the initial HMM topology file (HMM prototyoe).
- (d) the transcriptions file in MLF format.
- (e) the file of HMMs names list.

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

4.2 Recognition Phase

The recognition phase comprises the following steps:

1. Create the words dictionary and the network (language model) using:

```
Create_HTK-DicNet.sh ../SpanishNumbers/INF/samplesDB.dat \  
Dictionary Network
```

The first argument is the “samplesDB.dat” file containing around 20,000 phrases of Spanish numbers names. The second and third 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 from the samples in “samplesDB.dat”.

```
ngram-count -text ../SpanishNumbers/INF/samplesDB.dat \  
-lm LM.arpa -order 2
```

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
```

2. In a similar way, as it was done for the “train.lst” file, we create the test samples list “test.lst”:

```

for m in $(< ../SpanishNumbers/INF/list.test); do
  if [ -e feaDir/$m.fea ]; then
    echo feaDir/$m.fea;
  else echo "Error: feaDir/$m.fea does not exist." 1>&2;
  fi;
done > test.lst

```

3. Finally, we perform the proper recognition through the **HTK** command **HVite**. For reasons of time, we only perform the recognition of two samples (`feaDir/210341.fea` and `feaDir/210342.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.

```

HVite -A -T 1 -o ST -p -17 -s 50 -H hmm/hmm_32/Macros_hmm \
-l '*' -i res32.mlf -w Network Dictionary HMMsList \
feaDir/210341.fea feaDir/210342.fea

```

In this case, the recognition is carried out using the trained HMMs with 32 Gaussians per state (defined in `hmm/hmm_32/Macros_hmm` file) and the lexicon and language model defined respectively in the “Dictionary” and “Network” files. The arguments “-s 50” and “-p -17” 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. Each recognized hypothesis is stored in the “res32.mlf” file. To see them and their respective references:

```

cat res32.mlf
cat $HOME/SpanishNumbers/FRASES_TXT/{210341.txt,210342.txt}

```

As we observe, these results are not so good, since they were recognized using HMMs trained with few samples (we used `train_red.lst` instead of `train.lst`).

4.3 Evaluation Phase

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

1. Create the reference transcriptions file (in MLF format) “`SamplesRef.mlf`”:

```

Create_Test-MLF.sh ../SpanishNumbers/FRASES_TXT \
SamplesRef.mlf

```

2. Employing the command **HResults**:

```

HResults -I SamplesRef.mlf HMMsList res32.mlf

```

which outputs:

```

===== HTK Results Analysis =====
Date: Thu Nov 11 19:13:43 2010
Ref : SamplesRef.mlf
Rec : res32.mlf
----- Overall Results -----
SENT: %Correct=0.00 [H=0, S=2, N=2]
WORD: %Corr=55.56, Acc=55.56 [H=5, D=1, S=3, I=0, N=9]
=====

```

where $Acc=55.56$ is the accuracy rate corresponding to $WER=44.44\%$ (word error rate). The WER is defined by:

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

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 CATTI Simulation

5.1 Short Review

In the CATTI framework, the user is directly involved in the transcription process since he/she is responsible of validating and/or correcting the HTR output. The process starts when the system predicts an initial whole transcription of (some adequate segment of) the input image. Then, the user reads this prediction until finding (and correcting) an error. This generates a new, extended prefix (the previous validated prefix, plus the user amendments), which is used by the HTR system to attempt a new prediction hypothesis, thereby starting a new cycle that is repeated until a final correct transcription is achieved. An example of this process is shown in figure 2. It is worth nothing in this example that non-interactive post-editing would have required the user to correct *six* errors from the original recognized hypothesis whereas, using the interaction feedback, only *two* user-corrections (the grey text in the final transcription T) are necessary to get the final error-free transcription.

	x	<i>opposed the Government Bill which brought</i>					
STEP-0	p						
STEP-1	$\hat{s} \equiv \hat{w}$	append	this	Comment	Bill	in that	thought
	p'						
	κ	opposed					
STEP-2	p	opposed					
	\hat{s}		the	Government	Bill	in that	thought
	p'	opposed	the	Government	Bill		
FINAL	κ					which	
	p	opposed	the	Government	Bill	which	
	\hat{s}						brought
	p'	opposed	the	Government	Bill	which	brought
	κ						#
	$p \equiv T$	<i>opposed</i>	<i>the</i>	<i>Government</i>	<i>Bill</i>	<i>which</i>	<i>brought</i>
Post-Edition		<i>opposed</i>	<i>the</i>	<i>Government</i>	<i>Bill</i>	<i>which</i>	<i>brought</i>

Figure 2: Example of CATTI interaction. Initially the prefix p is empty, and the system proposes a complete transcription $\hat{s} \equiv \hat{w}$ of the input image x . In each iteration the user reads this transcription, accepting a prefix p' of it. Then, he or she types in some keystrokes, κ , to correct some words of the transcription provided by the system, thereby generating a new prefix p (the accepted one p' plus the text κ added by the user). At this point, the system will suggest a suitable continuation \hat{s} to this prefix p and this process is repeated until a complete and correct transcription of the input signal is reached. In the final transcription, T , the user-typed text is typeset in grey color.

Two possible implementations of the CATTI decoder are studied: “Dynamic” Language Modeling where on each step a linear LM that accounts for the fixed prefix is concatenated with a LM that accounts for $P(s|p)$; and a faster approach based on word-graphs derived from the initial Viterbi decoding. The first one is easily solved by the Viterbi algorithm, but the computational cost grows quadratically with the number on words, on the other hand, the second approximation is suboptimal but presents a linear computational cost. In the case of this practice, and due to the simplicity of the task, the first one approximation is taken.

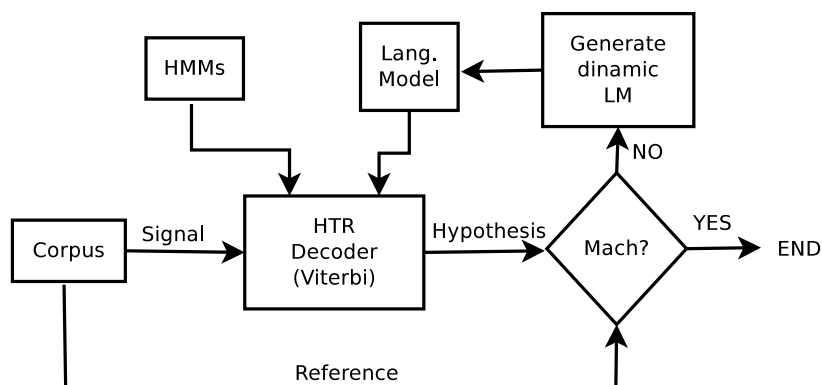


Figure 3: Scheme of the CATTI process where the process “Mach?” simulate the human actions

5.2 Simulation of CATTI user interaction

To make the experiments reproducible, and to avoid to the human large quantity of effort, the user actions nedded to correct the computer transcriptions must be simulated. As it is shown in the figure 3, the process “Mach?” receives the system transcription and its reference. It search for the first erroneous recognized word and correct it. Then a dynamic language model is generated. A new Viterbi parsing is done using the new dynamic language model. The process is repeated until the transcription has no errors.

The script “CATTI_simulation.sh” performs the this process. The tool **grammarTools** is used to generate dynamic language models given a prefix. Taking a valid prefix, for example “un millon cuarenta“ a dynamic language model can be build:

```
grammarTools -s "<s>" "</s>" -i Network -o DynamicLM \
-p "un millon cuarenta" -w
```

If the new language model is used to generate randomly sentences, it can be noted than all the sentences begins with the prefix used to generate it.

```
HSGen -n 100 DynamicLM Dictionary
```

5.3 Examples: CATTI on IAMDB Corpus

The “CATTI_IAMDB.log” file, placed into the directory “\$HOME/CATTI-Examples/”, holds the output of the shell-script “CATTI_simulation.sh” applied on the IAMDB corpus. To display this file correclty, be sure to set up the variable LESS to:

```
export LESS='-XR'
```

We recommend to perform this into xterm window,

```
xterm &
```

and when the command line prompt appears, run

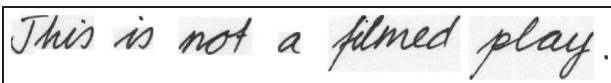
```
less ../CATTI-Examples/CATTI_IAMDB.log
```

Into the file “CATTI_IAMDB.log” there are several text blocks such as the bellow two examples.

Color codes: {

- Green text** denotes the reference text of the sample being transcribed.
- Red text:** denotes the user introduced/corrected words (by keyboard).
- Blue text:** denotes the error-free prefix validated by user before introducing/correcting the following word.
- Cyan text:** denotes the whole prefix (the validated part plus the corrected word) used by CATTI to suggest the new suffix continuation.
- Black text:** denotes the output hypothesis/suffix from CATTI.

Example 1



This is not a filmed play.

```
ID_Sample: c03-003d-s00
-----
REFERENCE: this is not a filmed play
HYPOTHESIS: this is not allowed play
ERRORS: D=1 S=1 I=0

Prefix 1: this is not a
HYPOTHESIS 1: this is not a filmed play

FINAL RESULT: this is not a filmed play
-----
```

The line “ERRORS: D=1 S=1 I=0” specifies the error types (D:deletions, S:substitutions and I:insertions) and number of them, between the reference and the original recognized hypothesis of the sample.

In this example, without any assistance the user would have to correct 2 errors (one word deletion: “a”, and one word substitution: “allowed” by “filmed”). However, using CATTI, just one iteration would be required to get the final error-free transcription. Note that, with the insertion of the word “a”, automatically is changed the next word (substitution error) from “allowed” to “filmed”.

Example 2

PRESIDENT KENNEDY renewed his pressure on Mr. Harold Macmillan to join the Common Market during their talks at Admiralty House, Whitehall, yesterday.

ID_Sample: a04-000-s00

REFERENCE: president kennedy renewed his pressure on mr harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

HYPOTHESIS: this per cent clay renewed his pressman is harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

ERRORS: D=1 S=4 I=2

Prefix 1: president

HYPOTHESIS 1: president kennedy renewed his pressman is harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

Prefix 2: president kennedy renewed his pressure

HYPOTHESIS 2: president kennedy renewed his pressure on the harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

Prefix 3: president kennedy renewed his pressure on mr

HYPOTHESIS 3: president kennedy renewed his pressure on mr harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

FINAL RESULT: president kennedy renewed his pressure on mr harold macmillan to join the common market during their talks at admiralty house whitehall yesterday

In the example above (larger than previous one) we have the original recognized hypothesis differing with respect to the reference in seven (post-editing) errors:

- One deletion: **mr**.
- Four substitutions: **cent** by **president**, **clay** by **Kennedy**, **pressman** by **pressure** and **is** by **on**,
- Two insertions: **this** and **per**.

Again, only four CATTI user-iterations were enough to get the final error-free transcription. In this example, the user have had to introduce three words:

1. **president** to replace the word **this**, causing CATTI automatically to remove the two next words **per cent** and change **clay** by **kennedy**.
2. **pressure** to replace **pressman**, causing CATTI to change **is** by **on**, although desafortunately insert after this the word **the**.
3. **mr** to replace **the**.

At the end of the file “CATTI_IAMDB.log” there is a table similar to this:

```
#####
Summary Results:
#####
```

#Er/Phr	#Samp	#Err	#Iter	Long-Ref	WER	WSR	EFR
*0	97	0	0	426	0.00	0.00	0.00
*1	55	55	58	232	23.71	25.00	-5.45
2	18	36	38	84	42.86	45.24	-5.56
3	11	33	26	57	57.89	45.61	21.21
4	3	12	7	14	85.71	50.00	41.67
5	1	5	3	6	83.33	50.00	40.00
6	2	12	4	8	150.00	50.00	66.67

GLOBAL: 18.50 16.44 11.11 (D=7 S=127 I=26)

GLOBAL: 57.99 46.15 20.41 (D=4 S=73 I=25) without considering *

where:

- #Er/Phr: number of errors per sentence.
- #Samp: number of sentences.
- #Err: number of errors.
- #Iter: number of iterations.
- Long-Ref: total number of words in the references.
- WER: word error rate.
- WSR: word stroke ratio or number of CATTI user-iterations to get the final transcription.
- EFR: estimated effort reduction, defined by:

$$EFR = \frac{WER - WSR}{WER} \cdot 100$$

5.4 CATTI Simulation Tool

The script named “CATTI_simulation.sh” is in charge to make a simulation of CATTI user interaction on a given corpus of phrases (features extraction files) and a language model. The implementation of it, is based on the recognition and evaluation phases described in section 4. The script can be run in the following way:

```
CATTI_simulation.sh test.lst hmm/hmm_2/Macros_hmm HMMsList \  
outDir 70 -250 Network Dictionary \  
SamplesRef.mlf |tee log
```

It takes 9 arguments:

1. the list of samples files to transcribe (`test.lst`).
2. the trained HMMs with 2 Gaussians per state (defined in `hmm/hmm_2/Macros_hmm`).
3. the file with the HMMs names list (`HMMsList`).
4. the output directory where will be stored the recognized transcriptions (`outDir`).
5. the grammar scale factor (70).
6. the word insertion penalty (-270).
7. the file with the model language to perform the recognition (`Network`).
8. the file with the vocabulary (`Dictionary`).
9. the file with the reference samples in MLF format (`SamplesRef.mlf`).

Its output is similar to the content of file “CATTI_IAMDB.log”.

6 Proposed Exercises

E-1 Analyze the two CATTI examples previously shown, and look for others similar into the file “CATTI_IAMDB.log”; i.e. other 5 examples where the user word corrections cause the CATTI changes automatically their following words.

E-2 Launch the script “CATTI_simulation.sh”, as is described in 5.4. In the summary results table located at the end of the output file “log”, there are entries for #Er/Phr for which the EFR is negative. Explain and justify what happens in such cases and give some examples taken from the “log” file.

Appendixes

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 [?]. 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 Recognition of a Complete 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 -p -17 -s 50 -S test.lst \
    -H hmm/hmm_32/Macros_hmm -l '*' -i res32.mlf \
    -w Network Dictionary HMMsList
```

In this case the recognition is carried out on the samples files listed in `test.lst` file, using the trained HMMs with 32 Gaussians per state (defined in `hmm/hmm_32/Macros_hmm` file) and the lexicon and language model defined respectively in the “Dictionary” and “Network” files. Each recognized hypothesis is stored in the “res32.mlf” file. The arguments “-s 50” and “-p -17” 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.