Automatic Speech Recognition 2012–13: Coursework lab (part 2 of 2)
— Continuous speech recognition: Speaker Adaptation —

Hiroshi Shimodaira and Steve Renals (*Revision* : 2.0)

In this lab-session we will

- adapt speaker-independent HMMs that we have trained in the previous sessions to target speaker's speech using MLLR
- investigate the performance of speaker adaptation with varying the number of regression classes.

1 Updating your environment

To make your experimental environment up-to-date, please copy two new python files to your work directory:

```
% cd  WorkDir [1]
% cp /afs/inf.ed.ac.uk/group/teaching/asr/ASR/python/HTK_MLLR.py .
% cp /afs/inf.ed.ac.uk/group/teaching/asr/ASR/python/mllrWSJCAM0.py .
```

As before, when you use the python functions you need to be in your *WorkDir*directory (and nowhere else); also you need to import the functions in python using

```
from HTK_WSJCAM0 import *
from HTK_MLLR import *
```

and to set the variables defined at the top of `trainWSJCAM0.py` and `mllrWSJCAM0.py`. The easiest way is probably to create your own version (e.g. `myMllrWSJCAM0.py`).

2 MLLR speaker adaptation

In this session, we focus on adaptation of mean vectors of Gaussian components only, and we will not consider adaptation of variance parameters. We assume supervised adaptation here, meaning that we know the utterances (i.e. phone sequences) of adaptation data for the target speaker.

Adapting a mean vector, $\boldsymbol{\mu}$, of a Gaussian distribution linearly can be formulated as

$$\begin{aligned} \hat{\boldsymbol{\mu}} &= A\boldsymbol{\mu} + \boldsymbol{b} = W\boldsymbol{\xi} \\ \boldsymbol{\xi} &= \left(1,\, \boldsymbol{\mu}^T\right)^T \\ W &= \left(\boldsymbol{b},\, A\right) \end{aligned}$$

where W is a transformation matrix.

In HTK, the adaptation procedure normally consists of the following steps:

Step 0 Prepare a set of speaker-independent HMMs and its "stats" file that contains state occupation statistics.

Step 1 Generate a regression-class tree.

Step 2 Given a set of adaptation data of a target speaker, estimate speaker's transformation matrix W for each regression class.

[1] `/afs/inf.ed.ac.uk/group/teaching/asr/Work/` *YourLoginName*

Step 3 Given a set of test(evaluation) data of the target speaker, adapt the HMMs to the speaker and carry out recognition.

Speaker adaptation experiments can be done using either monophone models (R1, R2) or tied-state triphone models (R6).

The following sections show examples of adapting the monophone models stored in `models/R2/hmm4`. Thus, you will need to modify the calls to the python functions according to the models you actually use (set `hmmType=tiedTriGMM`, `modelFile=tiedTriModelList` and `tiedTriphoneFlag=True` for tied triphones).

2.1 Step 0: preparation

Here, we assume that we are going to adapt `models/R2/hmm4`. First of all, make sure that you have those files:

models/R2/hmm4/MODELS	···	HMM parameter file
models/R2/hmm4/stats	···	state occupation statistics file

If you do not have the "stats" file above[2], then make a copy of the existing models file (for backup)

```
% mv models/R2/hmm4/MODELS models/R2/hmm4/MODELS.bak
```

Then call the python function `trainHMMs` to create it:

```
trainHMMs(hmmIter=3, n=1, hmmType=monoGMM, modelFile=spMonoModelList,
labelFile=realignedMonoMLF, statsFlag=True)
```

Make sure you have got the stats file now.

2.2 Step 1: generating regression class tree

The following python function will create a regression class tree with two regression classes.

```
createMllrRegClasses(nRegClasses=2, hmmIter=4, hmmType=monoGMM, modelFile=spMonoModelList)
```

This will create the following files in `models/R2/hmm4/classes`.

`global`	···	global transform
`regtree_2.tree`	···	regression class tree
`regtree_2.base`	···	list of base classes

(The digit "2" above represents the number of regression classes.) For details, see HTKBook[Section 3.6.2, 9.1.3, 9.1.4].

2.3 Step 2: estimate transformations

The python function `estimateMllrTransforms` calls HERest twice to estimate MLLR transformations using adaptation data set, "dev-01.scp".

Pass	transformation class	xform-file extension
1st-pass	global class	mllr1
2nd-pass	regression classes	mllr2

The adaptation set,"dev-01.scp", contains the same speakers as those of the test set, "si_dt5a-div3.scp", but utterances are mutually exclusive between the two sets. Calling the function

```
estimateMllrTransforms(nRegClasses=2, hmmIter=4, hmmType=monoGMM,
modelFile=spMonoModelList, tiedTriphoneFlag=False)
```

will create a directory

[2]You will have a stats file already for triphone models (R9), because it is needed for tree clustering. A stats file can be created with the "-s" option for HERest.

`models/R2/hmm4/xforms_2`

containing these transformation files for each speaker.

c31.mllr1, c31.mllr2,
c34.mllr1, c34.mllr2,
⋮ ⋮
c49.mllr1, c49.mllr2

2.4 Step 3: adaptation and recognition

The python function `recogniseSpeechMllr` is a modified version of `recogniseSpeech` used previously, which calls HVite in a way such that MLLR model adaptation is used for recognition. Transformation files for each speaker are automatically chosen according to the speech file name, which contains the speaker ID (this is done with the "-h" option).

(HVite is called with a lot of options. Try to understand the meaning of each option with consulting the HTKbook.)

Call the python function:

```
recogniseSpeechMllr(nRegClasses=2, hmmIter=4, hmmType=monoGMM,
modelFile=spMonoModelList, smallFlag=True, tiedTriphoneFlag=False)
```

which might take a while to finish. Make sure if the script (HVite) has not produced a warning message of "No tokens survived". If there was more than one such a message, you would need to change the pruning threshold (-t) of HVite. Use the small test set (`smallFlag=True`) because some of the files in the full test set (which are not included in the small test set) were used for the supervised adaptation.

To get recognition accuracy, run the python function `scoreWERMllr` (which is similar to `scoreWER`)

```
scoreWERMllr(hmmIter=4, hmmType=monoGMM, smallFlag=True)
```

3 Further study

There are a lot of issues that were not covered in the lab sessions. For example:

- Parameterising speech data (into MFCCs)
- Creating lists of training/testing sets
- Converting phone/word labels into HTK readable form
- Creating word (pronunciation) dictionaries
- Creating language models
- Recognition with a trigram word language model