Building a word based digit recognizer using HTK

Introduction

HTK (Hidden markov model toolkit) has been prepared by Cambridge University. It is an open source library and set of tools to perform speech recognition using HMM.
All source code is available, and it's ANSI C, that can be compiled on Windows or Unix.
It consists of two parts,
a- a library: that supports the basic functionality to perform HMM different algorithms, process speech signal and so on…
b- a set of tools: that are based on the library, and each tool is available (after compilation) as an exe that can be called from the command line. Each tool can perform a set of functions. E.g. HHed can edit the models files, HVite apply viterbi algorithm to perform recognition, HErest is used to train the HMM models. We will use these tools only.

To build a speech recognition system, we first need to train some HMM models, that will be used to decode the input speech. To train the models we will need to provide a training data, which is a set of speech files for each word (in the case of word based models). Then these models will be used to recognize the new speech.
In this project we will build a word based digit recognizer. The digit recognizer will be required to recognize Arabic digits from 0 to 9. A word based means that we will train an HMM model for each word, in this case we will need 10 models.

IMPORTANT WARNINGS: HTK is dumb, the files names and everything is case sensitive. Any file should end with an end line. Any error in files format will generate strange errors, so take care.

1- Data Preparation

In this step we will prepare the training data wav files models names. Also we will prepare the used grammar.

1.1 Recording wave file

The first step is to record speech data. So we will record each word at least 20 times. So we will have at least 200 wave files. Please make sure that all speech files have the format 16KHz sampling rate and 16 bits per sample mono (single channel). You can use wavesurfer for recording.
To make it easier follow a standard in files naming as X_YY.wav, where X is the digit, and YY is the file number for the current digit. E.g. for the digit 1, 1_01.wav, 1_02.wav and so on. Put all files in a folder named 'wav'.

1.2 Prepare the label file

To be able to train the models, we need to write what's in each wave file. This data is written in the master label file in the following format:

```
#!MLF!#
"*/0_00.lab"
```

```
SIL
ZERO
SIL
.
"*/0_01.lab"
SIL
ZERO
SIL
.
"*/0_02.lab"
SIL
ZERO
SIL
.
…
```

Write the previous lines for all recorded wave files for all digits and save it in the file trans.mlf.

1.3 Prepare the grammar

The grammar describes the allowed sequence of words. As we allow any number of consecutive digits, we will use a grammar file as follows:

```
$digit = WAHED | ETNEEN | TALAATA | ARBA3A | 5AMSA |
      SETTA | SAB3A | TAMANYA | TES3A | ZERO;
( SIL <$digit [SIL] > SIL )
```

The first 2 lines define a *variable* named $digit, that can be any one of the given words. The last line describes the grammar, where the brackets < > means any number of $digit words is allowed to come.
Save the grammar in a file named grammar.txt.
Then convert the grammar to a word network that can be used by HTK using the command:

```
HParse grammar.txt wdnet.txt
```

Which will generate the network file wdnet.txt.

1.4 Prepare the dictionary

The dictionary specifies the models that represent each word. The dictionary is meant for the case of phoneme based models. But we still have to write a dictionary in our case (the word based case). Each entry in the dictionary will contain the word and its model name (which can be the same). The dictionary can be as follows:

```
WAHED    WAHED
ETNEEN   ETNEEN
TALAATA  TALAATA
ARBA3A   ARBA3A
5AMSA    5AMSA
```

```
SETTA    SETTA
SAB3A    SAB3A
TAMANYA  TAMANYA
TES3A    TES3A
ZERO     ZERO
SIL      SIL
```
Save the dictionary in a file named: dictionary.txt

1.5 Extract Features

This step involves generating features files for each wave file. Features can be extracted using the HTK tool HCompv. The HCompv tool divides the utterance (a wave file is called utterence) into a number of frames, and gets a fixed number of features for each frame. The features we will use is MFCC (Mel frequency cepstral coefficients). We will use a feature vector size 13 (12 cepstral coefficient plus zeroth coefficient). Also at run time delta and delta-delta will be computed, totaling in 39 values as a feature vector.
To use HCompv we need a configuration file and a script file.
Configuration file conding_cfg.txt

```
# Coding parameters
SOURCEFORMAT = WAV
TARGETKIND = MFCC_0
TARGETRATE = 100000.0
SAVECOMPRESSED = T
SAVEWITHCRC = T
WINDOWSIZE = 250000.0
USEHAMMING = T
PREEMCOEF = 0.97
NUMCHANS = 26
CEPLIFTER = 22
NUMCEPS = 12
ENORMALISE = F
```

The script file will contain a line for each file mentioning the ource wave file and the target mfcc file, (we will put mfcc files in a folder named ‘mfcc’). Name the script file coding_scp.txt. Its format is as follows:

```
C:\digitRec\wav\0_00.wav C:\digitRec\mfcc\0_00.mfc
C:\digitRec\wav\0_01.wav C:\digitRec\mfcc\0_01.mfc
C:\digitRec\wav\0_02.wav C:\digitRec\mfcc\0_02.mfc
…
```

Then call HCompv using the command
```
HCopy -T 1 -C coding_cfg.txt -S coding_scp.txt
```

2- Models Building and Training

In this step we will create HMM models and train them.

2.1 Create the prototype file

To describe an HMM model, we have to specify the number of states in a model, mean and variance of each feature, and transition matrix.
In our case for easiness, we will use 10 states for each model. The prototype model can described as follows:

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "proto"
<BeginHMM>
<NumStates> 12
<State> 2
<Mean> 39
0.0 0.0 0.0 ...
<Variance> 39
1.0 1.0 1.0 ...
<State> 3
<Mean> 39
0.0 0.0 0.0 ...
<Variance> 39
1.0 1.0 1.0 ...
<State> 4
<Mean> 39
0.0 0.0 0.0 ...
<Variance> 39
1.0 1.0 1.0 ...
… (Put here the other states from state 2 to state 11)
<TransP> 12
0.0 1.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.6 0.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.6 0.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.6 0.4 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.6 0.4 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.6 0.4 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.4 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.4 0.0 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.4 0.0 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.4 0.0
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.6 0.4
0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

Here we define 12 states because we want 10 states, and start and end states. The start and end states are used to connect models to each other. At each state we have 39 mean values and 39 variance values (13*3). Save the proto as ‘proto’ (The file name should match the model name).

2.2 Compute global parameters

In this step we will compute the global mean and variance and set all of the Gaussians in a given HMM to have the same mean and variance. We will use the tool HComV.

First prepare a list file of all mfcc files to be used as the training script, name it train_scp.txt. It should have format similar to:

```
C:\digitRec\wav\0_00.mfc
C:\digitRec\wav\0_01.mfc
…
```

Then we use the following command (Note: create the folder hmm0 before calling the command):
HCompV -C train_cfg.txt -f 0.01 -m -S train_scp.txt -M hmm0 proto

Here train_cfg.txt is similar to coding_cfg.txt except we remove the line "SOURCEFORMAT = WAV" and set "TARGETKIND = MFCC_0_D_A" To force HTK to compute delta and acceleration parameters at run time.

The previous command will scan the given set of files, and create a new hmm file in the directory hmm0 containing the updated means and variances, also it will create the file 'vFloors' for variance flooring.

2.3 Create Master HMM file

The master HMM file (MMF file) is the file that contains all HMM models. In our case, we have 10 models. The MMF file consists of a set of macros, each macro starts with the symbol ~ then the macro name, e.g. ~o for options, ~h for models … and a macro ends by the next macro. We will create it by the following steps:

1- Create a text file named hmm.txt
2- Copy the macro ~o from hmm0\proto
3- Copy the ~v macro from hmm0\vFloors
4- Copy the ~h macro 11 times, one for each model. And replace proto by each model name.
5- For the SIL model, leave only the first 3 states (make number of states 5) and transition matrix 5*5

2.4 Train the HMM models

Training of HMM models is done by applying Baum Welch algorithm on the training files provided. We use the tool HERest. We can use the following command (You have to create the folder hmm1 first):

```
HERest -C train_cfg.txt -I trans.mlf -t 250.0 150.0 1000.0 -S
train_scp.txt -H hmm0/hmm.txt -M hmm1 ModelsList.txt
```

The command will train HMM models and save the result in the folder hmm1. IT takes the configursation file train_cfg.txt and the transcription file (prepared in step 1.2), and train_scp.txt file, and the models list ModelsList.txt. ModelsList.txt contains a list of all models names one name on each file.

We need to run HERest a number of iterations. So run the previous command more 3 times, but make it read the models from hmm1 instead of hmm0 as follows:

```
HERest -C train_cfg.txt -I trans.mlf -t 250.0 150.0 1000.0 -S
train_scp.txt -H hmm1/hmm.txt -M hmm1 ModelsList.txt
```

3- Running the recognizer

To do speech recognition using the trained models, you can use HVite tool, which uses viterbi algorithm to decode the input speech.

First prepare the configuration file run_cfg.txt as follows:

```
# Waveform capture
SOURCERATE=625.0
SOURCEKIND=HAUDIO
SOURCEFORMAT=HTK
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
USESILDET=T
MEASURESIL=F
```

The use the command

```
HVite -H hmm1/hmm.txt -C run_cfg.txt -w wdnet.txt -p 0.0 -s 5.0
dictionary.txt ModelsList.txt
```