A Real-time Continuous Speech Recognition System in English

EE627 Course Project 2013

Group Members:

1. Abhishek Jindal (10024)
2. Ankan Bansal (10095)
3. Asapu Shiva (10152)
4. Ayush Jain (10180)
5. Prenit Wankhede (10530)

Contents

1 Objective

The objective of this project is to build a continous speech recognition system for English. A real time speech recognitiion needs to be built using the TIMIT database. Experimental results for speech recognition in terms of word error rate (WER) also need to be provided.

2 Methodology for Speech Recognition

The methodology followed for performing the recogniton experiments are as follows:

- Building the task grammar (a "language model")
- Constructing a dictionary for the models
- Creating transcription files for training data
- Encoding the data (feature processing)
- (Re-)training the acoustic models
- Evaluating the recognizer against the test data
- Reporting recognition results

3 Database for Training

The database used for training is TIMIT database. The TIMIT corpus of read speech is designed to provide speech data for acoustic-phonetic studies and for the development and evaluation of automatic speech recognition systems. TIMIT contains broadband recordings of 630 speakers of eight major dialects of American English, each reading ten phonetically rich sentences. The TIMIT corpus includes time-aligned orthographic, phonetic and word transcriptions as well as a 16-bit, 16kHz speech waveform file for each utterance. Corpus design was a joint effort among the Massachusetts Institute of Technology (MIT), SRI International (SRI) and Texas Instruments, Inc. (TI). The speech was recorded at TI, transcribed at MIT and verified and prepared for CD-ROM production by the National Institute of Standards and Technology (NIST). The TIMIT corpus transcriptions have been hand verified. Test and training subsets, balanced for phonetic and dialectal coverage, are specified. Tabular computer-searchable information is included as well as written documentation.

4 Tools Used

The tools used to implement this project is the HTK Toolkit. HTK is the Hidden Markov Model Toolkit developed by the Cambridge University Engineering Department (CUED). This toolkit aims at building and manipulating **Hidden Markov Models** (HMMs). HTK is primarily used for speech recognition research (but HMMs have a lot of other possible applications) HTK consists of a set of library modules and tools available in C source form.

5 Some Theoretical Details

5.1 Hidden Markov Models (HMMs)

A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. A HMM can be considered the simplest dynamic Bayesian network. In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a *hidden* Markov model, the state is not directly visible, but output, dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore the sequence of tokens generated by an HMM gives some information about the sequence of states. A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

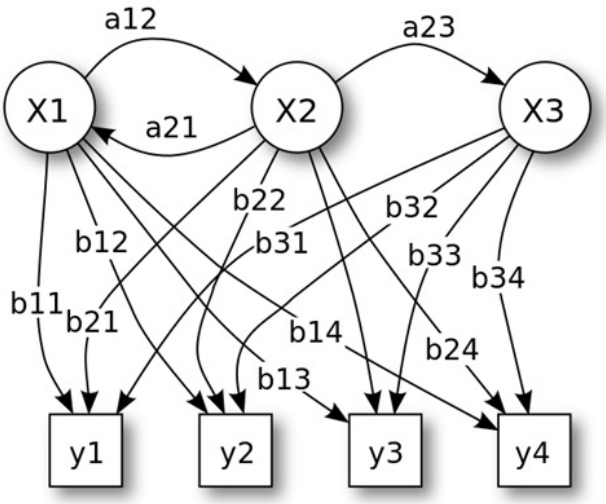

Figure 1: HMM Architecture

Figure 1 shows the basic architecture of a HMM. Here, x are the states, y are the possible observations, a are the state transition probabilities and b are the output probabilities.

5.2 HMMs for Speech Processing

The architecture of HMMs for speech processing is similar to that shown in figure 1. For speech processing x represent the speech code (phoneme/word), y are the feature vectors, a are the speech code transition probabilities and b represent the feature vector probabilities.

6 Fundamentals of HTK

HTK is a toolkit for building HMMs. HMMs can be used to model any time series and the core of HTK is similarly general-purpose. However, HTK is primarily designed for building

HMM-based speech processing tools, in particular recognisers. Thus, much of the infrastructure support in HTK is dedicated to this task. As shown in figure 2, there are two major processing stages involved. Firstly, the HTK training tools are used to estimate the parameters of a set of HMMs using training utterances and their associated transcriptions. Secondly, unknown utterances are transcribed using the HTK recognition tools.

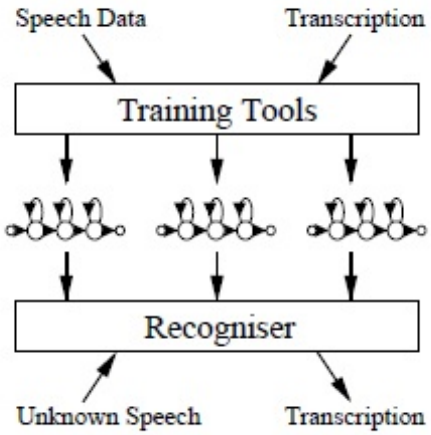

Figure 2: Processing Stages

Figure 3 represents the basic working architecture of the HTK toolkit. It shows the basic functions that are used and at which stages they are used.

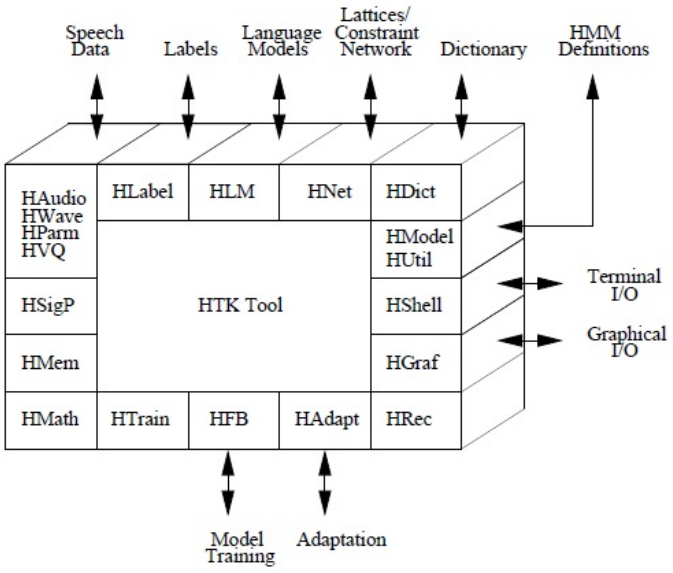

Figure 3: Processing Stages

7 Processing Stages

7.1 Task Grammar

HTK provides a grammar definition language for specifying simple task grammars.

7.2 Dictionary

The first step in building a dictionary is to create a sorted list of the required words. It is necessary to build a word list from the sample sentences present in the training data. To build robust acoustic models, it is necessary to train them on a large set of sentences containing many words and preferably phonetically balanced.

7.3 Recording the Data

7.4 Transcription Files

To train a set of HMMs, every file of training data must have an associated phone level transcription. This is mostly hand labelled data. In the TIMIT dataset transcription files for all the sentences have been provided.

7.5 Coding the Data

The speech recognition tools cannot process directly on speech waveforms. These have to be represented in a more compact and efficient way. Here Mel Frequency Cepstral Coefficients (MFCCs), which are derived from FFT-based log spectra, are used. The steps for calculating the MFCC components can be listed as:

- Take the Fourier Transform of (a windowed excerpt of) a signal
- Map the powers of the spectrum obtained above onto the mel scale, using triangular overlapping filter bank
- Take the logs of the powers at each of the mel frequencies
- Take the discrete cosine transform of the list of mel log powers, as if it were a signal
- The MFCCs are the amplitudes of the resulting spectrum

In our case, we have used 39 features:

- 13 MFCC coefficients
- 13 Delta coefficients which are the first derivatives of the MFCC coefficients
- 13 Acceleration coefficients - the second derivatives of the MFCC coefficients

The conversion from the original waveform to a series of acoustical vectors is done with the HCopy HTK tool. It requires a configuration file (config) which specifies all the conversion parameters. It also uses a list of all source files and their corresponding output files. Files containing such lists are called script files. For example for creating the feature vectors for the training data, we have used the command: *HCopy -T 1 -C configTIMIT -S codetr.scp*
Figure 4 shows the architecture of the HCopy function.

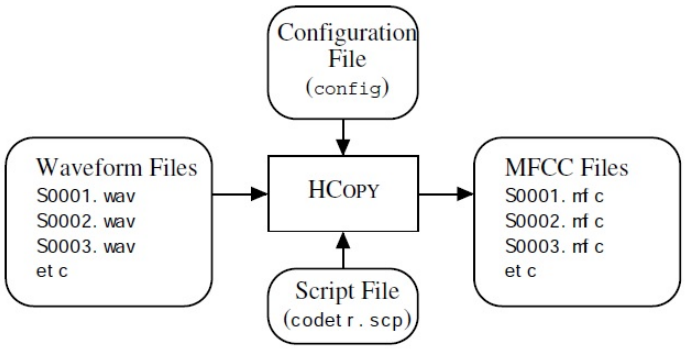

Figure 4: HCopy

7.6 Configuration

The mean and variance of each state was set to zero and one respectively for each of the 39 dimensions. The HMM for each phoneme was initialized to have 5 hidden states.
Source Rate = 625
Target Rate = 100000
Window size = 250000

7.7 HMM Definitions

The principle function of HTK is to manipulate sets of hidden Markov models (HMMs). The definition of a HMM must specify the model topology, the transition parameters and the output distribution parameters. The HMM observation vectors can be divided into multiple independent data streams and each stream can have its own weight. In addition, a HMM can have ancillary information such as duration parameters. HTK supports both continuous mixture densities and discrete distributions. HTK also provides a generalised tying mechanism which allows parameters to be shared within and between models. In order to encompass this rich variety of HMM types within a single framework, HTK uses a formal language to define HMMs.

The a priori topology for each HMM is first chosen:

- number of states
- form of the observation functions (associated with each state)
- disposition of transitions between states

In HTK, a HMM is described in a text description file (proto file).

7.8 HMM Training

The next step is to estimate the parameters of the HMMs from examples of the data sequences that they are intended to model. This process of parameter estimation is usually called *training*. HTK supplies four basic tools for parameter estimation: HCompV, HInit, HRest and HERest. HCompV and HInit are used for initialisation. HCompV will set the mean and variance of

every Gaussian component in a HMM definition to be equal to the global mean and variance of the speech training data. This is typically used as an initialisation stage for flat-start training. HRest and HERest are used to refine the parameters of existing HMMs using Baum-Welch Re-estimation.

Where there is limited training data and recognition in adverse noise environments is needed, so-called fixed variance models can offer improved robustness. These are models in which all the variances are set equal to the global speech variance and never subsequently re-estimated. The tool HCompV can be used to compute this global variance.

Although HTK gives full support for building whole-word HMM systems, the bulk of its facilities are focussed on building sub-word systems in which the basic units are the individual sounds of the language called phones. One HMM is constructed for each such phone and continuous speech is recognised by joining the phones together to make any required vocabulary using a pronunciation dictionary. Figure 5 shows the basic procedures involved in training a subword model.

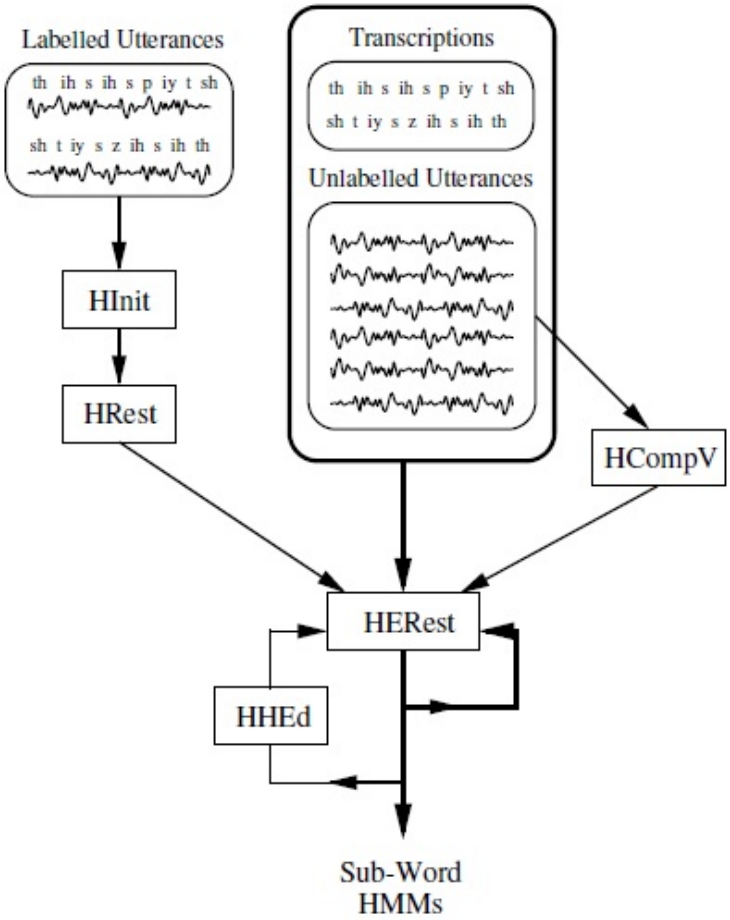

Figure 5: Training Subword HMMs

HCompV is used to assign the global speech mean and variance to every Gaussian distribution in every phone HMM. This so-called *flat start* procedure implies that during the first cycle of embedded re-estimation, each training utterance will be uniformly segmented. The hope then is that enough of the phone models align with actual realisations of that phone so that on the second and subsequent iterations, the models align as intended.

The core process involves the embedded training tool HERest. HERest uses continuously

spoken utterances as its source of training data and simultaneously re-estimates the complete set of subword HMMs. For each input utterance, HERest needs a transcription i.e. a list of the phones in that utterance. HERest then joins together all of the subword HMMs corresponding to this phone list to make a single composite HMM. This composite HMM is used to collect the necessary statistics for the re-estimation. When all of the training utterances have been processed, the total set of accumulated statistics are used to re-estimate the parameters of all of the phone HMMs.

The commands used by us are
HCompV -T 1 -C config -f 0.01 -m -S train2.scp -M model/hmm0 proto
and
HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S train.scp -H model/hmm', num2str(i-1), '/macros -H model/hmm', num2str(i-1), '/hmmdefs -M model/hmm', num2str(i), ' monophones0'.
Details are in the code (Appendix 1).

7.9 Recognition

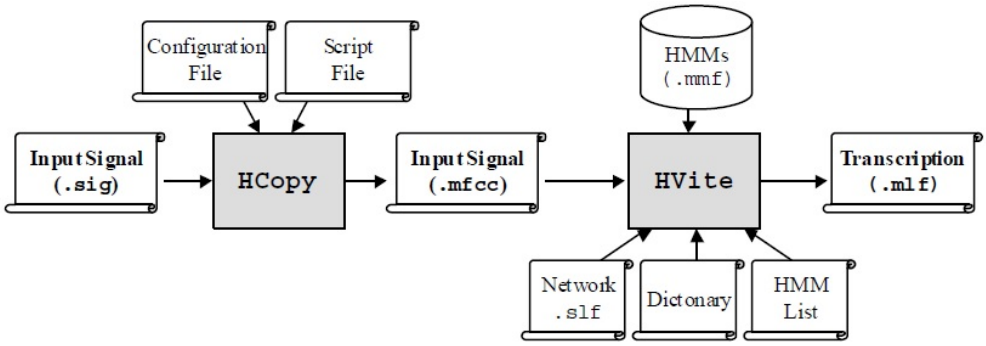

Figure 6: Recognition process of an unknown input signal

The recognition process can be summarised as:

- An input speech signal input.sig is first transformed into a series of "acoustical vectors (here MFCCs) with tool HCopy, in the same way as what was done with the training data (Acoustical Analysis step). The result is stored in an input.mfcc file (often called the *acoustical observation*).
- The input observation is then process by a Viterbi algorithm, which matches it against the recognisers Markov models. This is done by tool HVite.

7.10 Error Rates

The ref.mlf and rec.mlf transcriptions are compared with the HTK performance evaluation tool, HResults.

8 Results

Results show an above 74 percent recognition rate (figure 7)

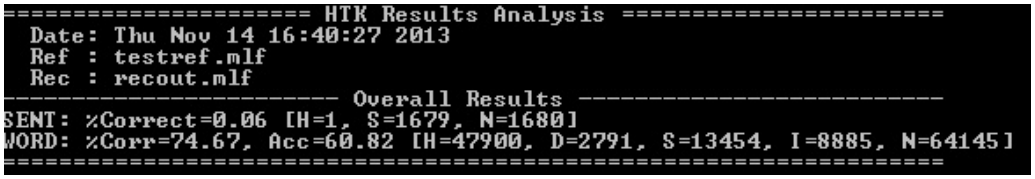

Figure 7: Results

References

[1] Rabiner, L. R., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," Proceedings of the IEEE, vol. 77, no. 2, Feb. 1989.

[2] Juang, B. H., and Rabiner, L. R., "Hidden Markov models for speech recognition," Technometrics , 1991

[3] http://en.wikipedia.org/wiki/TIMIT

[4] Garofolo, J. S., and Lamel, L. F., "TIMIT Acoustic-Phonetic Continuous Speech Corpus"

[5] Hultzen, I. S., Allen Jr., J.H.D., and Miron, M. S., "Tables of Transitional Frequencies of English Phonemes," 1964

[6] Lamel, L. F., Kassel, R. H., and Seneff, S., "Speech database development: Design and analysis of the acoustic-phonetic corpus," 1986

[7] Lee, K. F., and Hon, H. W., "Speaker-independent phone recognition using Hidden Markov Models" 1989

[8] Leung, H. C., "A procedure for automatic alignment of phonetic transcriptions with continuous speech," 1985

9 Appendix: MATLAB Code

```
close all, clc;

htkdir = ['C:\Users\Ankan\Desktop\NEW\Acads\SEM 7\EE627\htk\'];
homedir = ['C:\Users\Ankan\Desktop\NEW\Acads\SEM 7\EE627\TIMIT - Copy\'];
traindir = ['C:\Users\Ankan\Desktop\NEW\Acads\SEM 7\EE627\TIMIT - Copy\TRAIN\'];
testdir = ['C:\Users\Ankan\Desktop\NEW\Acads\SEM 7\EE627\TIMIT - Copy\TEST\'];


eval(['!']);
eval(['!mkdir label']);
eval(['!mkdir mfcc']);
eval(['!mkdir model']);

for i=0:23
    eval(['!mkdir model\hmm', num2str(i)])
end
```

```
%% Make a gram file and rune HParse
disp('make a gram file');

fid = fopen('gram', 'w');
fprintf(fid, '%s\n', ['$beginend = h#;']);
fprintf(fid, '%s', ['$phone = bcl | b | dcl | d | gcl | g | pcl | p | tcl |'...
    't | kcl | k | ']);
fprintf(fid, '%s', ['dx | q | jh | ch | s | sh | z | zh | f | th | v | dh |'...
    ' m | n | ng | em | en | eng | ']);
fprintf(fid, '%s', ['nx | l | r | w | y | hh | hv | el | ']);
fprintf(fid, '%s', ['iy | ih | eh | ey | ae | aa | aw | ay | ah | ao | oy |'...
    ' ow | ']);
fprintf(fid, '%s\n', ['uh | uw | ux | er | ax | ix | axr | ax-h | pau | epi;']);
fprintf(fid, '%s\n', ['($beginend <$phone> $beginend)']);
fclose(fid);

%input('Press enter to continue');
eval(['!HParse -T 1 gram wdnet']);

disp('Make a monophones0 file. The file contains phone symobls');

fid = fopen('monophones0', 'w');
    fprintf(fid, '%s\n', 'b');
    fprintf(fid, '%s\n', 'd');
    fprintf(fid, '%s\n', 'g');
    fprintf(fid, '%s\n', 'p');
    fprintf(fid, '%s\n', 't');
    fprintf(fid, '%s\n', 'k');
    fprintf(fid, '%s\n', 'dx');
    fprintf(fid, '%s\n', 'q');
    fprintf(fid, '%s\n', 'jh');
    fprintf(fid, '%s\n', 'ch');
    fprintf(fid, '%s\n', 's');
    fprintf(fid, '%s\n', 'sh');
    fprintf(fid, '%s\n', 'z');
    fprintf(fid, '%s\n', 'zh');
    fprintf(fid, '%s\n', 'f');
    fprintf(fid, '%s\n', 'th');
    fprintf(fid, '%s\n', 'v');
    fprintf(fid, '%s\n', 'dh');
    fprintf(fid, '%s\n', 'm');
    fprintf(fid, '%s\n', 'n');
    fprintf(fid, '%s\n', 'ng');
    fprintf(fid, '%s\n', 'em');
    fprintf(fid, '%s\n', 'en');
    fprintf(fid, '%s\n', 'eng');
    fprintf(fid, '%s\n', 'nx');
    fprintf(fid, '%s\n', 'l');
    fprintf(fid, '%s\n', 'r');
    fprintf(fid, '%s\n', 'w');
    fprintf(fid, '%s\n', 'y');
    fprintf(fid, '%s\n', 'hh');
    fprintf(fid, '%s\n', 'hv');
    fprintf(fid, '%s\n', 'el');
    fprintf(fid, '%s\n', 'iy');
    fprintf(fid, '%s\n', 'ih');
    fprintf(fid, '%s\n', 'eh');
    fprintf(fid, '%s\n', 'ey');
```

```
    fprintf(fid, '%s\n', 'ae');
    fprintf(fid, '%s\n', 'aa');
    fprintf(fid, '%s\n', 'aw');
    fprintf(fid, '%s\n', 'ay');
    fprintf(fid, '%s\n', 'ah');
    fprintf(fid, '%s\n', 'ao');
    fprintf(fid, '%s\n', 'oy');
    fprintf(fid, '%s\n', 'ow');
    fprintf(fid, '%s\n', 'uh');
    fprintf(fid, '%s\n', 'uw');
    fprintf(fid, '%s\n', 'ux');
    fprintf(fid, '%s\n', 'er');
    fprintf(fid, '%s\n', 'ax');
    fprintf(fid, '%s\n', 'ix');
    fprintf(fid, '%s\n', 'axr');
    fprintf(fid, '%s\n', 'ax-h');
    fprintf(fid, '%s\n', 'bcl');
    fprintf(fid, '%s\n', 'dcl');
    fprintf(fid, '%s\n', 'gcl');
    fprintf(fid, '%s\n', 'pcl');
    fprintf(fid, '%s\n', 'tcl');
    fprintf(fid, '%s\n', 'kcl');
    fprintf(fid, '%s\n', 'pau');
    fprintf(fid, '%s\n', 'epi');
    fprintf(fid, '%s\n', 'h#');
fclose(fid)


%% PREPARING TRAINING DATA %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid1 = fopen('codetr.scp', 'w');
fid2 = fopen('train.scp', 'w');
%fid3 = fopen('fname.praat', 'w');

for n0=1:8
    D = dir([traindir, 'DR', num2str(n0)]);
    for n1=3:size(D,1)
        D2 = dir([traindir, 'DR', num2str(n0),'\',D(n1).name,'\*.WAV']);
        D3 = dir([traindir, 'DR', num2str(n0),'\',D(n1).name, '\*.PHN']);
        for n2 = 1:size(D2,1)
            filename = [traindir, 'DR', num2str(n0), '\', D(n1).name, '\'...
                D2(n2).name];

%             fprintf(fid3, '%s\n', ['Read from file... ', filename]);
%             fprintf(fid3, '%s\n', ['Write to WAV file... ', filename]);
%             fprintf(fid3, '%s\n', 'Remove');

            handdefname = [traindir, 'DR', num2str(n0), '\',D(n1).name, '\'...
                D3(n2).name];

            newfname=D2(n2).name;
            newfname=[newfname(1:end-4) '_tr.mfc'];
            mfcfname = [homedir, 'mfcc\', 'DR', num2str(n0), '_', D(n1).name,...
                '_', newfname];

            fprintf(fid1, '%s\n', ['"', filename, '"', ' ', '"', mfcfname,...
                '"']);
```

```
            fprintf(fid2, '%s\n', ['"', mfcfname, '"']);

            newlname = D3(n2).name;
            newlname = [newlname(1:end-4) '_tr.lab'];
            labfname = [homedir, 'label\', 'DR', num2str(n0), '_', D(n1).name,...
                '_', newlname];

            eval(['!copy ', '"', handdefname, '"', ' ', '"', labfname, '"']);

        end

    end
end
fclose(fid1);
fclose(fid2);
% fclose(fid3);

% disp('The original TIMIT wav files need to be converted to MSWAVE file format');
% disp('Make a praat script');
% input('Press enter to continue');

%% PREPARE TIMIT configuration file & run HCopy

fid = fopen('configTIMIT', 'w');
fprintf(fid, '%s\n', ['SOURCEKIND = ANON']);
fprintf(fid, '%s\n', ['SOURCEFORMAT = ALIEN']);
fprintf(fid, '%s\n', ['HEADERSIZE = 12']);
fprintf(fid, '%s\n', ['SOURCERATE = 625']);
fprintf(fid, '%s\n', ['TARGETKIND = MFCC_0_D_Z_A']);
fprintf(fid, '%s\n', ['TARGETRATE = 100000.0']);
fprintf(fid, '%s\n', ['WINDOWSIZE = 250000.0']);
fprintf(fid, '%s\n', ['USEHAMMING = T']);
fprintf(fid, '%s\n', ['PREEMCOEF = 0.97']);
fprintf(fid, '%s\n', ['NUMCHANS = 20']);
fprintf(fid, '%s\n', ['CEPLIFTER = 22']);
fprintf(fid, '%s\n', ['NUMCEPS = 12']);
fclose(fid);

eval(['!HCopy -T 1 -C configTIMIT -S codetr.scp']);
% eval(['!HCopy -T 1 -C configTIMIT -S train.scp']);


%% PREPARING TESTING DATA %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid1 = fopen('codete.scp', 'w');
fid2 = fopen('test.scp', 'w');
%fid3 = fopen('test.praat', 'w');

for n0 = 1:8

    D = dir([testdir, 'DR', num2str(n0)]);
    for n1 = 3:size(D, 1)

        D2 = dir([testdir, 'DR', num2str(n0), '\', D(n1).name, '\*.wav']);
        D3 = dir([testdir, 'DR', num2str(n0), '\', D(n1).name, '\*.phn']);

        %#%disp(D2(n1).name);
```

```
        for n2 = 1:size(D2, 1)
            filename=[testdir, 'DR', num2str(n0), '\', D(n1).name, '\'...
                D2(n2).name];

%             fprintf(fid3, '%s\n', ['Read from file... ', filename]);
%             fprintf(fid3, '%s\n', ['Write to WAV file... ', filename]);
%             fprintf(fid3, '%s\n', 'Remove');

            handdefname = [testdir, 'DR', num2str(n0), '\', D(n1).name, '\'...
                D3(n2).name];

            newfname = D2(n2).name;
            newfname=[newfname(1:end-4) '_te.mfc'];
            mfcfname = [homedir, 'mfcc\', 'DR', num2str(n0), '_', D(n1).name,...
                '_', newfname];

            fprintf(fid1, '%s\n', ['"', filename, '"', ' ', '"', mfcfname,...
                '"']);
            fprintf(fid2, '%s\n', ['"', mfcfname, '"']);

            newlname = D3(n2).name;
            newlname = [newlname(1:end-4) '_te.lab'];
            labfname = [homedir, 'label\', 'DR', num2str(n0), '_', D(n1).name,...
                '_', newlname];

            eval(['!copy ', '"', handdefname, '"', ' ', '"', labfname, '"']);
        end
    end
end

fclose(fid1);
fclose(fid2);
% fclose(fid3);

eval(['!HCopy -T 1 -C configTIMIT -S codete.scp']);

%% PREPARE configuration file
fid = fopen('config', 'w');
fprintf(fid, '%s\n', ['TARGETKIND = MFCC_0_D_Z_A']);
fprintf(fid, '%s\n', ['TARGETRATE = 100000.0']);
fprintf(fid, '%s\n', ['WINDOWSIZE = 250000.0']);
fprintf(fid, '%s\n', ['USEHAMMING = T']);
fprintf(fid, '%s\n', ['PREEMCOEF = 0.97']);
fprintf(fid, '%s\n', ['NUMCHANS = 20']);
fprintf(fid, '%s\n', ['CEPLIFTER = 22']);
fprintf(fid, '%s\n', ['NUMCEPS = 12']);
fclose(fid)

%%
disp('Make proto file');

fid = fopen('proto', 'w');
    fprintf(fid, '%s\n', ['~o <VecSize> 39 <MFCC_0_D_Z_A>']);
    fprintf(fid, '%s\n', ['~h "proto"']);
    fprintf(fid, '%s\n', ['<BeginHMM>']);
    fprintf(fid, '\t%s\n', ['<NumStates> 5']);
    fprintf(fid, '\t%s\n', ['<State> 2']);
    fprintf(fid, '\t\t%s\n', ['<Mean> 39']);
```

```
    fprintf(fid, '\t\t\t%s\n', ['0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 '...
        '0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0']);
    fprintf(fid, '\t\t%s\n', ['<Variance> 39']);
    fprintf(fid, '\t\t\t%s\n', ['1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 '...
        '1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1']);
    fprintf(fid, '\t%s\n', ['<State> 3']);
    fprintf(fid, '\t\t%s\n', ['<Mean> 39']);
    fprintf(fid, '\t\t\t%s\n', ['0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 '...
        '0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0']);
    fprintf(fid, '\t\t%s\n', ['<Variance> 39']);
    fprintf(fid, '\t\t\t%s\n', ['1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 '...
        '1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1']);
    fprintf(fid, '\t%s\n', ['<State> 4']);
    fprintf(fid, '\t\t%s\n', ['<Mean> 39']);
    fprintf(fid, '\t\t\t%s\n', ['0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 '...
        '0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0']);
    fprintf(fid, '\t\t%s\n', ['<Variance> 39']);
    fprintf(fid, '\t\t\t%s\n', ['1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 '...
        '1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1']);
    fprintf(fid, '\t%s\n', ['<TransP> 5']);
    fprintf(fid, '\t\t%s\n', ['0 1 0 0 0']);
    fprintf(fid, '\t\t%s\n', ['0 0.6 0.4 0 0']);
    fprintf(fid, '\t\t%s\n', ['0 0 0.6 0.4 0']);
    fprintf(fid, '\t\t%s\n', ['0 0 0 0.7 0.3']);
    fprintf(fid, '\t\t%s\n', ['0 0 0 0 0']);
    fprintf(fid, '%s\n', ['<EndHMM>']);
fclose(fid)

eval(['!HCompV -T 1 -C config -f 0.01 -m -S train2.scp -M model/hmm0 proto']);

fid = fopen('model/hmm0/proto', 'r');
F = fread(fid);
S = char(F');
SHMM = S( strfind(upper(S), '<BEGINHMM>') :end);
S1st3 =S(1: strfind(S, '~h') -1);
fclose(fid);

fid = fopen('model/hmm0/vFloors', 'r');
F = fread(fid);
SvFloors = char(F');
fclose(fid);

fid = fopen('model/hmm0/macros', 'w');
fprintf(fid, S1st3);
fprintf(fid, SvFloors);
fclose(fid);

fid1 = fopen('monophones0', 'r');
fid2 = fopen('model/hmm0/hmmdefs', 'w');
fid3 = fopen('dict', 'w');
fid4 = fopen('monophones1', 'w');

%%
while 1
    tline = fgetl(fid1);
    disp(tline);
    if ~ischar(tline)
        break;
```

```
    end

    fprintf(fid2, ['~h "', tline, '"\n']);
    fprintf(fid2, SHMM);
    fprintf(fid2, '\n');

    fprintf(fid3, [tline, ' ', tline, '\n']);

    fprintf(fid4, [tline, '\n']);
end

fprintf(fid4,['!ENTER\n']);
fprintf(fid4,['!EXIT\n']);

fprintf(fid3,['!ENTER []\n']);
fprintf(fid3,['!EXIT []\n']);

fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);

% input('Press enter to continue');

%%
fid1=fopen('phones0.mlf','w');
fid3=fopen('HLStatslist','w');

fprintf(fid1,'%s\n',['#!MLF!#']);
D=dir(['label/*tr.lab']);
for n=1:size(D,1)
    fprintf(fid1,'%s\n',['"*/',D(n).name,'"']);
    fprintf(fid3,[D(n).name,'\n']);

    fid2=fopen(['label/',D(n).name],'r');
    while 1
        tline=fgetl(fid2);
        if ~ischar(tline)
            break;
        end;
        if (tline(1)=='#')|(tline(1)=='"')
            fprintf(fid1,'%s\n',tline);
        else
            Tmat=sscanf(tline,'%d %d %s');
            Tstring=[char(Tmat(3:end))]';
            fprintf(fid1,'%s\n',Tstring);
        end
    end
    fprintf(fid1,'%s\n','.');
    fclose(fid2);
end
fprintf(fid1,'\n');
fclose(fid1);
fclose(fid3);

% input('Press enter to continue');

eval(['!HLStats -T 1 -b bigfn -o -I phones0.mlf monophones0 -S HLStatslist']);
```

```
eval(['!HBuild -T 1 -n bigfn monophones1 outLatFile']);

%%
fid1 = fopen('testref.mlf','w');
fprintf(fid1, '%s\n', ['#!MLF!#']);
D = dir(['label/*te.lab']);
for n = 1:size(D, 1)
    fprintf(fid1, '%s\n', ['"*\', D(n).name, '"']);
    fid2 = fopen(['label\', D(n).name], 'r');
    while 1
        tline = fgetl(fid2);
        if ~ischar(tline)
            break;
        end
        if (tline(1) == '#') | (tline(1) == '"')
            fprintf(fid1, '%s\n', tline);
        else
            Tmat=sscanf(tline,'%d %d %s');
            Tstring=[char(Tmat(3:end))]';
            fprintf(fid1,'%s\n',Tstring);
        end
    end
    fprintf(fid1, '%s\n', '.');
    fclose(fid2);
end
fprintf(fid1, '\n');
fclose(fid1);

%%
for i = 1:3
    eval(['!HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S '...
        'train.scp -H model/hmm', num2str(i-1), '/macros -H model/hmm',...
        num2str(i-1),'/hmmdefs -M model/hmm', num2str(i), ' monophones0']);
end

%%
fid = fopen('sil.hed', 'w');
fprintf(fid, ['AT 2 4 0.2 {pau.transP}\n']);
fprintf(fid, ['AT 4 2 0.2 {pau.transP}\n']);
fprintf(fid, ['AT 2 4 0.2 {h#.transP}\n']);
fprintf(fid, ['AT 4 2 0.2 {h#.transP}\n']);
fclose(fid);

eval(['!HHEd -T 1 -H model/hmm3/macros -H model/hmm3/hmmdefs -M model/hmm4 '...
    'sil.hed monophones0']);

%%
for i = 5:7
    eval(['!HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S '...
        'train.scp -H model/hmm', num2str(i-1), '/macros -H model/hmm',...
        num2str(i-1), '/hmmdefs -M model/hmm', num2str(i), ' monophones0']);
end

fid = fopen('MU2.hed', 'w');
fprintf(fid, ['MU 2 {*.state[2-4].mix}\n']);
fclose(fid);

%%
```

```
eval(['!HHEd.exe -T 1 -H model/hmm7/macros -H model/hmm7/hmmdefs -M model/hmm8 '...
    'MU2.hed monophones0']);

for i=9:11
    eval(['!HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S '...
        'train.scp -H model/hmm',num2str(i-1),'/macros -H model/hmm',...
        num2str(i-1),'/hmmdefs -M model/hmm',num2str(i),' monophones0']);
end

fid=fopen('MU4.hed','w');
fprintf(fid,['MU 4 {*.state[2-4].mix}\n']);
fclose(fid);

%%
%input('Press enter to continue');
eval(['!HHEd -T 1 -H model/hmm11/macros -H model/hmm11/hmmdefs -M model/hmm12 '...
    'MU4.hed monophones0']);

% input('Press enter to continue');
 for i=13:15
     eval(['!HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S '...
         'train.scp -H model/hmm',num2str(i-1),'/macros -H model/hmm',...
         num2str(i-1),'/hmmdefs -M model/hmm',num2str(i),' monophones0']);
 end

fid=fopen('MU8.hed','w');
fprintf(fid,['MU 8 {*.state[2-4].mix}\n']);
fclose(fid);

%%
eval(['!HHEd -T 1 -H model/hmm15/macros -H model/hmm15/hmmdefs -M model/hmm16 '...
    'MU8.hed monophones0']);

for i=17:23
    eval(['!HERest -T 1 -C config -I phones0.mlf -t 250.0 150.0 1000.0 -S '...
        'train.scp -H model/hmm',num2str(i-1),'/macros -H model/hmm',...
        num2str(i-1),'/hmmdefs -M model/hmm',num2str(i),' monophones0']);
end

%%
eval(['!HVite -T 1 -H model/hmm23/macros -H model/hmm23/hmmdefs -S test.scp '...
    '-i recout.mlf -w wdnet -p 0.0 -s 5.0 dict monophones0']);
%input('Press enter to continue');
disp('With bigram language model:');
eval(['!HVite -T 1 -H model/hmm23/macros -H model/hmm23/hmmdefs -S test.scp -i '...
    'recout_bigram.mlf -w outLatFile -p 0.0 -s 5.0 dict monophones0']);

%%

eval(['!HResults -T 1 -I testref.mlf monophones0 recout.mlf']);
eval(['!HResults -T 1 -e n en -e aa ao -e ah ax-h -e ah ax -e ih ix -e l el -e '...
    'sh zh -e uw ux -e er axr -e m em -e n nx -e ng eng -e hh hv -e pau pcl -e '...
    'pau tcl -e pau kcl -e pau q -e pau bcl -e pau dcl -e pau gcl -e pau epi '...
    '-e pau h# -I testref.mlf monophones0 recout.mlf']);
```