Voiced/Unvoiced and Silent Classification Using HMM Classifier based on Wavelet Packets BTE features

Amr M. Gody[1]
Fayoum University

Abstract

Wavelet Packets Best Tree Encoded (BTE) features is used here as base features for HMM classifier. The research aimed to introduce the newly designed features that are discussed in [1]. The considered problem is Voiced, Unvoiced and Silent classification. Comparison to the 19 filter banks features is provided. Although it is simple and straight forward, BTE makes comparable results to the 19 elements features vector based on filters bank. A very accurate hand labeled database called SCRIBE is used. Voiced sounds are recognized in 81% success rate. Silent periods are detected in 84.5% success rate. The unvoiced sounds are not recognized using the proposed features. It gives a 5.5% success rate. This low rate of unvoiced detection affects the overall performance. The overall performance of 64.5% is achieved. This overall performance is expected to be dramatically changed in case of adding some unvoiced attributes to BTE.

1. Introduction

Using good features is the key of accurate speech recognizer. Recognizer's success depends on three main factors. The first factor is the database used in the training phase. The second factor is the features used to train the model. The third factor is the mathematical model used to recognize the different classes in the speech signal.

BTE features are discussed in [1]. BTE inherits some human attributes by considering the human hearing mechanism in processing the received speech. Received speech's stream is classified into logarithmic bands before it is being processed by human brain [1]. This human nature is described by Mel scale as shown in figure 1.

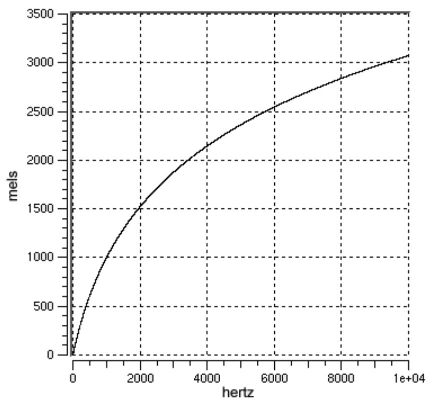


Figure 1: Mel scale curve that models the human hearing response to different frequencies [2].

[1] Department of Electrical Engineering, Email: amg00@fayoum.edu.eg

Mel frequency reflects what human can discriminate. It is a scale that reflects what human can hear. As shown in figure 1, from 4000(HZ) to 8000 (HZ) only 1000 (Mel) change while from 0(HZ) to 1000 (HZ) a 1000(Mel) change is appeared. The curve in low frequency till 1000(HZ) indicates that the human ear can be highly discriminative. This property starts to be degraded toward the higher frequencies. As shown in figure 1, change after 4000(Hz) in frequency tends to make almost very low change in Mel scale. This phenomenon indicates that human ear is much sensitive to low frequencies than to high frequencies. It is expected that most of the information contained into speech is located in the low frequency area of the total bandwidth of speech signal. Recognizers based on Filter banks tries to satisfy this logarithmic relation that was explained by figure 1. It is not wise to handle the frequency band in a linear manner while it is not like that in human hearing mechanism.

The objective of this paper is to test BTE through a comparative study. This research is a preliminary work to introduce Wavelet Packets Best Tree 4 point Encoded (BTE) features. The database is selected to be very accurate hand labeled database. SCRIB[2] database is selected. SCRIBE consists of a mixture of read speech and spontaneous speech. The read speech material consists of sentences selected from a set of 200 "phonetically rich" sentences and 460 "phonetically compact" sentences and a two-minute continuous passage. Hidden Markov Model (HMM) is chosen as the mathematical model of the speech recognizer. This model is chosen for its good reputation in speech recognition domain. The model is implemented using The Hidden Markov Model Toolkit HTK[3]. Signal processing, system evaluation and results preparation are calculated using Speech Filling System SFS[4]. All programming and logic are made using Microsoft C Sharp[5] (C#).

Voiced, Unvoiced and silent are three main classes in any spoken language. Almost all phonetics is either Voiced or unvoiced. Silent periods are those periods where no speech exists. The detection of unvoiced speech in the presence of additive background noise is complicated by the fact that unvoiced speech is very similar to white noise [3]. The problem of detecting unvoiced appears in this preliminary research. It is almost confused in equal proportion between Voiced and silent.

2. Framework

In this section the framework of this research will be fully explained. The system has many actors/ resources.

1. SFS platform.
2. HTK platform.
3. Batch and Cue Logic platform (BCL). Microsoft C# is used to implement BCL platform.
4. Hand labeled Database (SCRIBE).
5. Matlab platform.

[2] SCRIBE database: www.phon.ucl.ac.uk/resource/scribe
[3] Hidden Markov Model Toolkit HTK: http://htk.eng.cam.ac.uk/
[4] Speech Filling System SFS: http://www.phon.ucl.ac.uk/resource/sfs/
[5] Microsoft C# is one of the programming languages by Microsoft Corporation. C# implements Object Oriented Programming (OOP). It bears both simplicity and advanced programming technique. It is considered the number one language nowadays. For more information go to http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

Step 1 [Creating SFS files]

BCL is used to import all sound files provided by the SCRIBE into SFS formatted files. Also BCL is used to import all corresponding label files into the same SFS files. After this step each SFS file contains waveform item and annotation item.

Step 2 [Mapping phonetic labels into Voiced, Unvoiced and Silent labels]

A new map file is constructed. The map file contains the map for each phonetic symbol into one of the three classes {VOI, UNV and SIL}. Symbols are VOI for Voiced sound, UNV for Unvoiced sound and SIL for Silent or pauses. SFS is used to apply the map to SFS file. Then BCL is used to apply the SFS map to all SFS files.

Step 3 [Preparing two different groups for two parallel experiments]

SFS files are cloned into two sets. This is to use each SET into different experiment. SET_A will be used in VOC19 feature experiment and SET_B will be used in BTE features experiment.

Step 4 [Apply feature extraction function on all SFS files]

SET_A: SFS is used to apply VOC19 to the available samples.

SET_B: Matlab is used to extract BTE features. BCL is used to apply the Matlab function to all available samples. Then finally SFS is used to import all feature vectors into the SFS files.

Step 5 [HMM preparation]

SET_A: Three HMM models are prepared. Each model is 3 states. HTK is used to initialize each model based on training samples, label files and feature vector files.

SET_B: The same process as in SET_A is followed.

Step 6 [Training HMM models]

In both sets, HTK is used to train the available HMM models. The training depends on the feature vector files, label files and selected training files list. Training continues till a convergence in log probability happened for each model.

Step 7 [Testing HMM models against test files]

Test files are some SFS files that were never being used in the training phase. HTK is used to test HMM models against the selected test files in both groups. HTK generates label file for each test file. Each label file is imported using BCL to the corresponding SFS file. This will cause that each SFS test file contain two annotations. One is the reference annotation generated in step 2 and the other one is the test annotation.

Step 8 [Evaluation]

Each SFS test file will be analyzed using SFS. A confusion matrix is generated for each SFS test file. Results are registered for both groups. Then results are tabulated and graphs are obtained to view and compare the results.

3. Database

SCRIBE database is used in this research. It is multi-speakers database. Each file is phonetically transcribed and segmented.

The following commands invoke SFS to add Sound file and the corresponding annotation file into SFS formatted file. This SFS file will act as a container for all items {Speech file, features data and annotation data}

```
Slink -i1.01 -f SamplingRate  Sound_file  sfsfile
Anload -S Annotation_file sfsfile
```

To apply the same command to all the available samples, BCL is used. The following is the program written to do the function. All speech files in SCRIB are listed into a string array called "files". Speech files in SCRIBE have an extension called "PES". Speech is sampled at 20000 (HZ). All annotation files in SCRIBE have an extension called "PEA". The Annotation file is located in the same folder as the corresponding speech file in SCRIBE database. All functionalities for SFS are packaged into a class library called "SPLib[6]". A certain class for processing SFS commands is implemented. It is called "SFSFile". It is located into "SPLib".

```
foreach (string file in files)
        {
          SPLib.SFSFile f = new SPLib.SFSFile();
          string sfsFile;
          string anFile;
          int start = file.LastIndexOf('\\');
          int end = file.LastIndexOf('.');
          sfsFile = targetdir + file.Substring(start, end - start) + ".sfs";
          anFile = file.Substring(0, file.Length - 1) + "a";
          f.open(sfsFile);
          f.AddSPItem(file, 20000);
          f.AddANItem(anFile);
         }
```

"AddSPItem" and "AddANItem" are subroutines that contain the SFS commands which are previously listed.

Now we have SFS file for each of the database files. This SFS file will act as a container for speech signal, associated annotation symbols and associated features.

The next operation to be applied on the speech database files is to map the phonetic annotations into Voiced, Unvoiced and Silent annotations. This is important for HTK to understand the classes to be trained. Let us denote the new annotation set with a suitable name for the upcoming references. The new set is called speech type set (STS). So, STS contains three speech type symbols

1. Voiced speech symbol (VOI).
2. Unvoiced speech symbol (UNV).
3. Silent periods or no speech symbol (SIL).

The first step is to make a MAP file that links each phonetic symbol into a suitable type symbol in STS. This file is manually created (by human not by program). Part of the map file is shown below:

```
#      SIL
##     SIL
%tc    SIL
+      SIL
/      SIL
3:     VOI
3:?    VOI
3:a    UNV
3:af   UNV
3:f    UNV
3:~    VOI
=l     VOI
=lx    VOI
=lx?   VOI
=lxf   UNV
```

The first column is the phonetic symbol and the second column is the map to STS symbol. A complete version of the map file may be downloaded from [4]. To apply

[6] SPLIB: Speech lib class library. It is a C# class library by Amr M. Gody to work with SFS and Matlab. It encapsulates all needed logic and business to work with speech signal using SFS or Matlab.

the map operation to annotation item inside an SFS file, the following command line is invoked:

```
Anmap -m  mapfile sfsfile
```

To apply the map operation on all the available SFS files, the following program is written as BCL:

```
foreach (string file in files)
            {
                SPLib.SFSFile f = new SPLib.SFSFile();
                f.open(file);
                f.MapAnnotation(mapfile);
            }
```

All SFS files are listed into string array called "files". Then for each "file" in "files" the map annotation is applied. " `MapAnnotation`" is a subroutine contains the SFS command that was indicated above.

4. Features extraction

In this section the process of feature extraction will be explained. We have two different groups as indicated in section 2. SET_A will be designated for VOC19 while SET_B will be chosen for BTE features. SFS will be used to apply VOC19 on all available samples. The following is the SFS command to do the function:

```
voc19 sfsfile
```

To apply the above command to all available samples in SET_A, the following program is written as BCL:

```
foreach (string file in files)
            {
                SPLib.SFSFile f = new SPLib.SFSFile(file);
                f.VOC19();
            }
```

All SFS files in SET_A are listed into string array called "files". Then the loop is applied on each file into files. VOC19 subroutine contains the SFS command that was indicated above. After this step, each SFS file in SET_A contains a new item that express VOC19 features. It is called coefficients item.

Now the coefficients item, contained into the SFS file that represents VOC19 features, needs to be exported into an HTK formatted file to be used in further step during the training of HMM model using HTK. The following SFS command do this function:

```
Colist -H sfsfile
```

To apply the above SFS command on all the available SFS files into SET_A, the following code snippet in BCL is used:

```
foreach (string file in files)
            {
                SPLib.SFSFile f = new SPLib.SFSFile();
                f.open(file);
                f.Co2HTK();

            }
```

In the above code snippet, all SFS files in SET_A are listed into a string array called "files". The function "Co2HTK" contains the SFS command needed to export the features from the SFS file to HTK formatted file.

It is also needed to extract the annotation item from the SFS file to an HTK formatted annotation file. This is achieved by calling the following SFS command:

```
anlist -h -O sfsfile
```

Then BCL is used to apply the function on all the available SFS files in SET_A and SET_B. The following code snippet in BCL is used to achieve this objective:

```
foreach (string file in files)
            {
                SPLib.SFSFile f = new SPLib.SFSFile();
                f.open(file);
                f.An2HTK();
            }
```

In the above code snippet, all SFS files in SET_A and in SET_B are listed into string array called "files" Then the function "An2HTK" is called for each file in the string array. "An2HTK" contains the SFS command mentioned above.

A parallel process is implemented on SFS files in SET_B. This time the proposed features (BTE) will be extracted. Matlab instead of SFS is used to implement BTE features extraction process. The following code snippet is the core part of Matlab function to implement BTE features extraction.

```
function [res] = BTE (frame, depth)
    nbIn = nargin;
    nbout = nargout;
    if nbIn < 1 ,    error('Not enough input arguments.');
    elseif nbIn == 1,     level = 4;
    elseif nbIn == 2,     level = depth;
    end;
    if nbout < 1 , error('Not enough output arguments.'); end;
    t = wpdec(frame,level,'db4','shannon');
    u = leaves (t);
    bt =  besttree(t);
    v = leaves (bt);
    res = box4encoder(v);
end
```

The function "box4encoder" in the above code snippet is responsible for encoding Best tree as indicated in [1].

To apply BTE algorithm on all available samples in SET_B, BCL is used. The following code snippet is used to apply BTE to all available speech samples assigned for SET_B experiment.

```
foreach (string file in files)
            {
                SPLib.SFSFile f = new SPLib.SFSFile();
                f.open(file);
                f.ExportWAV();
                string wfile = file.Substring(0, file.Length - 3) + "WAV";
                mat.wav2bte (1, wfile);

            }
```

All SFS files in SET_B are listed into a string array called files. The speech waveform is exported from the SFS file to a known format called WAV file format. This is important to pass the sound file to the Matlab function. The Matlab function called "wav2bte" is called for each file in the string array "files". For more information on the function "wav2bte" you may referee to [1].

Now it is needed to prepare the generated BTE files for being used by HTK. BCL is used to write such a converter. The following cod snippet is written for the converter function:

```
public static void BTEtoHTK(string BTEfile, string htkfile)
        {

BTEfile f1 = new BTEfile(BTEfile);
HTKFile f2 = new HTKFile();
int samplein100ns = Convert.ToInt32 (  f1.SampleLength * 1e-3 / 100e-9);
short bytesperhtksample =Convert.ToInt16 (  f1.BytesPerSample * 4 /
f1.BytesPerElemnt); // HTK is 4 bytes/element
f2.create(f1.NumberOfSamples, samplein100ns, bytesperhtksample,
SPLib.HTKParamKind.USER, htkfile);
            int n;
            n = f1.NumberOfSamples;
            int m = f1.ElementsPerSample;
            for (int i = 0; i < n; i++)
            {


                for (int j = 0; j < m; j++)
                {
                    int elm =(int) f1.ReadInt16();
                    f2.write( elm);
                }
            }
            f1.close();
            f2.close();
        }
```

By the end of the above step, we should have all the training files needed by HMM for both groups as shown in table3.

Table 1: Snapshot of HMM training files generated so far by the end of feature extraction step.

Group	Feature type	HTK feature files	HTK annotation files
SET_A (24 files)	VOC19	AAPA0001.dat ⋮	AAPA0001.lab ⋮
SET_B (24 files)	BTE	AAPA0001.htk ⋮	AAPA0001.lab ⋮

5. Training HMM

After features extraction step, it is the time for testing the features into a pattern recognition process. As indicated in Table 1, two sets of files are prepared. They are both ready for training HMM models using HTK.

First step in this phase is to design HMM model that best fit the information needed to be recognized. The model here is 3 states left to right model. This assumes that the recognized pattern is assumed to have three different parts. The parts are consequent parts. The first part is the left one and the last part is the right one. This assumption is very close to the reality as the consecutive sounds is supposed to have a transition periods at the boundaries and a stable period at the middle. Figure 14 explains the relation between the proposed design model and speech sound. In this experiment there are there sounds to be recognized {Voiced (VOI), Unvoiced (UNV) and silent (SIL)}.

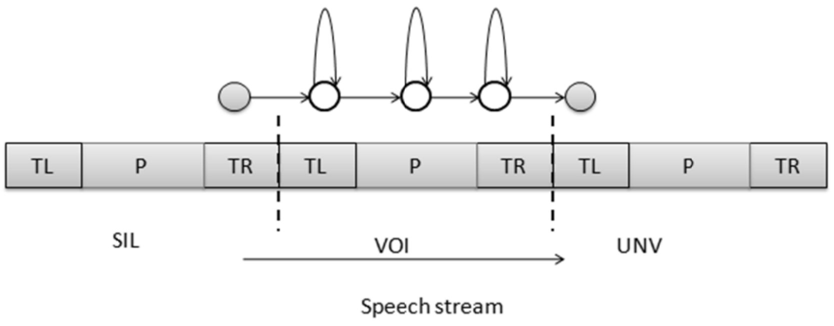


Figure 2: Relation between HMM model and speech sounds to be recognized. TL is the leading transition period and TR is the trailing transition period while P is the stable phone period.

The model contains two non emitting states which appear in gray color in figure 2. The non emitting states are important in HTK to indicate the entry and the exit points to the model. Gaussian Probability Distribution Function (PDF) is used in each state to fit the variability of the sound. To define an HMM model for HTK the following script is written into a separate text file.

```
~o
<STREAMINFO> 1 19
<VECSIZE> 19<NULLD><FBANK><DIAGC>
~h "SIL"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<MEAN> 19
 1.404372e+001 1.146923e+001 1.089870e+001 7.190041e+000 2.423316e+000 1.263892e+000
1.634800e+000 4.447996e-002 1.654768e+000 3.511177e+000 4.378909e+000 3.915090e+000
8.674143e-002 1.574287e-001 -6.058807e-001 -1.295122e+000 -2.223198e+000 -
2.120687e+000 -9.183334e-001
<VARIANCE> 19
 3.086359e+002 2.563524e+002 1.796036e+002 1.263492e+002 8.962412e+001 7.586296e+001
7.740655e+001 6.912480e+001 8.007733e+001 9.697153e+001 1.057107e+002 1.151898e+002
7.569676e+001 7.488948e+001 6.804313e+001 6.226783e+001 5.014687e+001 5.277541e+001
6.593863e+001
<GCONST> 1.210665e+002
<STATE> 3
<MEAN> 19
 -1.550593e+000 -3.269745e+000 -4.287612e+000 -5.243945e+000 -5.796925e+000 -
5.822775e+000 -5.803507e+000 -5.910261e+000 -5.803027e+000 -5.761147e+000 -
5.671880e+000 -5.702754e+000 -5.948490e+000 -5.959568e+000 -5.960902e+000 -
5.970729e+000 -5.993655e+000 -5.991790e+000 -5.961835e+000
<VARIANCE> 19
 8.319610e+001 4.678292e+001 2.388155e+001 8.794478e+000 2.526003e+000 1.837515e+000
2.220891e+000 1.482371e+000 2.225448e+000 2.601058e+000 3.406267e+000 3.234430e+000
8.082389e-001 8.287914e-001 8.249093e-001 5.735081e-001 2.694195e-001 3.057849e-001
6.422817e-001
<GCONST> 5.132733e+001
<STATE> 4
<MEAN> 19
 4.351817e+000 2.717550e+000 2.140117e+000 1.390050e+000 -1.326494e+000 -1.220001e+000
4.477349e-001 -1.336653e+000 -1.657292e-002 9.020020e-001 2.071208e+000 3.250645e+000
-2.780049e-001 -8.483851e-001 -9.663507e-001 -9.924042e-001 -1.579547e+000 -
1.328159e+000 -4.995179e-001
```

```
<VARIANCE> 19
 1.096481e+002 9.111847e+001 8.397858e+001 7.596156e+001 5.026793e+001 4.964570e+001
7.590692e+001 6.081587e+001 6.890288e+001 7.740077e+001 8.851939e+001 1.061187e+002
7.655155e+001 7.167074e+001 7.540655e+001 8.126243e+001 7.148788e+001 7.071327e+001
8.899091e+001
<GCONST> 1.172263e+002
<TRANSP> 5
 0.000000e+000 1.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
 0.000000e+000 7.436733e-001 2.563267e-001 0.000000e+000 0.000000e+000
 0.000000e+000 0.000000e+000 9.159696e-001 8.403045e-002 0.000000e+000
 0.000000e+000 0.000000e+000 0.000000e+000 7.777685e-001 2.222315e-001
 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000 0.000000e+000
<ENDHMM>
```

The above script defines all model parameters. It defines the following items

1- Number of states.
2- Feature type.
3- Transition Matrix.
4- Gaussian PDF parameters in each state (Means and Variance).
5- The name of the class. In the above example it is "SIL".

For more details you may referee to [5].

The above is an initial definition. This definition will be adapted on the vision of the training data available for the training phase. Two similar HMM models will be defined to model VOI and UNV sounds.

Now it is important to split the available speech database files into two groups. One group will be used for training and the other group will be used for test. The following list is the training set files. It is saved into a text file called "train.lst".

```
AAPA0002.dat
AAPA0003.dat
AAPA0004.dat
ACPA0002.dat
ACPA0003.dat
ACPA0004.dat
AEPA0002.dat
AEPA0003.dat
AEPA0004.dat
AFPA0002.dat
AFPA0003.dat
AFPA0004.dat
AHPA0002.dat
AHPA0003.dat
AHPA0004.dat
AMPA0002.dat
AMPA0003.dat
AMPA0004.dat
```

Another list will be prepared for test. The following is the test list. It is saved into a text file called "TEST.LST".

```
AAPA0001.dat
ACPA0001.dat
AEPA0001.dat
AFPA0001.dat
AHPA0001.dat
AMPA0001.dat
```

We have two sets of files for testing the model as indicated in section 4. SET_A for the filter banks features and SET_B for BTE features. It is important to configure HTK such that it can understand the type of features under test. The following text is saved into a text file called "config.txt".

```
# config.txt - HTK basic parameters
SOURCEFORMAT = HTK
TARGETKIND = FBANK
NATURALREADORDER = T
```

The configuration file will be provided for any HTK command to configure it to correctly understand the provided features during the test or the training phases. The above is the configuration file for Filter banks features. The following is the configuration file for BTE features type.

```
# config.txt - HTK basic parameters
SOURCEFORMAT = HTK
TARGETKIND = USER
NATURALREADORDER = T
```

BTE is a new feature type so that it should be provided to HTK as user defined type as indicated in the above script "TARGETKIND = USER".

After defining the three HMM models, it is better to initialize them using the avalible database. This is good before starting the training phase. The following HTK command is used to intitialize each HMM model:

```
hinit -T 1 -c config.txt -s train.lst SIL
hinit -T 1 -c config.txt -s train.lst VOI
hinit -T 1 -c config.txt -s train.lst UNV
```

The above three commands should initialize the available three HMM models on the vision of the available database for each class. The above step will be applied on the initial models in {SET_A and SET_B}.

Now the models are ready to be trained using the HTK. The following commands are used to train the models in both groups:

```
HRest -T 1 -C config.txt -S train.lst -l VOI VOI
HRest -T 1 -C config.txt -S train.lst -l UNV UNV
HRest -T 1 -C config.txt -S train.lst -l SIL SIL
```

The above step may be repeated till log probability approaches to 0 or approaches to stable value. As soon as the model is well trained, we can start the testing phase. By the end of this step we should have 6 HMM models as indicated in table 2.

Table 2: HMM files after the training phase.

Group	Feature type	HMM files
SET_A (3 files)	VOC19 (Filter Banks)	VOI UNV SIL
SET_B (3 files)	BTE	VOI UNV SIL

6. Testing HMM

As shown in table 2, we have three HMM models for each group of files. Now it is needed to test the trained models using the available testing files in each group. First it is needed to prepare dictionary and word net. Dictionary contains all recognized words while the Word net contains the grammar. Both of them are important for HTK to get it correctly functioning. The problem we address in this research is a simple classification problem that may not need grammars. So the word net will be prepared in such way that matches with our needs. Figure 3 explains the way HTK alters grammars. The top in the hierarchy is the word. Each word may be expressed in a network of phones as each phone network represents certain pronunciation for the associated word. And finally each phone is expressed in HMM model.

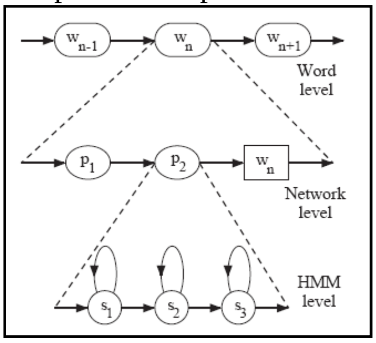


Figure 3: HTK recognizer in depth. The abbreviations are explained as W for Word, P for Phone and S for state. The dotted boundary explains the decomposition of the root element[5].

In our case the network is very simple. It will be constructed statistically from the available samples. Each sample has a label file that explains sample contents in term of VOI, UNV and SIL symbols. The following is a part of certain label file:

```
24917000 26727500 UNV
26727500 30867000 VOI
30867000 32193500 UNV
32193500 32375000 SIL
32375000 32414500 VOI
```

The first column indicates the beginning of the segment and the second column indicates the end of the segment. Numbers are in term of 100(ns). For example 24917000 means segment (UNV) will start at$24917000 \times 100 \times 10^{-9} = 2.4917(sec)$. The following HTK command is invoked to build the word net from the available label files.

```
HBuild voices.dic voices.net
```

The above command uses the symbols in the file "voices.dic" to construct the file "voices.net" using all label files exist in the same directory. The file "voices.dic" may be like the following script:

```
SIL [SIL] SIL
VOI [VOI] VOI
UNV [UNV] UNV
```

The dictionary file maps the word to its possible pronunciation phone streams. Here in our example the word is the same as the phone stream. Word VOI is constructed of phone VOI. In our experiment the word is the same as the phone. It is a one level recognition. We do not have further resolutions for each word. The word between the square brackets is the output symbol. It is used by HTK to provide suitable output when word is recognized. It is an optional parameter. The network file generated by "HBuild" command is like the following:

```
 VERSION=1.0
N=7    L=9
I=0    W=!NULL
I=1    W=!NULL
I=2    W=UNV
I=3    W=SIL
I=4    W=VOI
I=5    W=!NULL
I=6    W=!NULL
J=0     S=0    E=1    l=0.00
J=1     S=5    E=1    l=0.00
J=2     S=1    E=2    l=-1.10
J=3     S=1    E=3    l=-1.10
J=4     S=1    E=4    l=-1.10
J=5     S=2    E=5    l=0.00
J=6     S=3    E=5    l=0.00
J=7     S=4    E=5    l=0.00
J=8     S=5    E=6    l=0.00
```

J means joint, S means start, E means end, W means word symbol, l means log probability and I is node identifier. Figure 4 explains the structure of the word net generated by "HBuild".

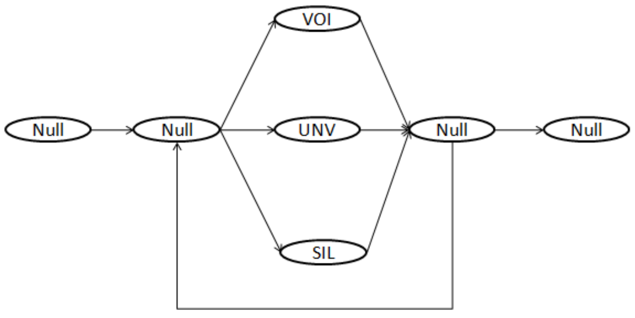


Figure 4: Word net structure.

After constructing the dictionary and word net files, it is possible to start testing the models. All test files will be fed to HTK for the recognition process. The following HTK command is used to start testing the models against the available testing files:

```
HVite -T 1 -C config.txt -w voices.net -o S -S test.lst voices.dic words.lst
```

The above HTK command should be applied to both sets {SET_A and SET_B}. The file "words.lst" contains the name of HMM model files. In this experiment it is like the following script:

```
SIL
VOI
UNV
```

After executing the above command, all recognition results will be exported into files in the same name as the test files with a new file extension ".rec". The generated ".rec" file is just similar to the standard label file. The following is a part of such generated files:

```
11800000 13600000 VOI
13600000 30000000 SIL
30000000 31200000 VOI
```

```
31200000 32800000 SIL
32800000 36000000 VOI
36000000 37400000 SIL
```

To start results mining process, it is required to import the recognition files into the associated SFS files. BCL will be used to apply the import process on all files in both sets {SET_A and SET_B}. The following code snippet is written to make the function:

```
foreach (string file in files)
            {
                int index = file.LastIndexOf('.');
                string sfsfile = file.Substring(0, index) + ".sfs";
                Process a = new Process();
                a.StartInfo.FileName = "anload";
                a.StartInfo.Arguments = "-h "+file +" " + sfsfile;
                a.StartInfo.RedirectStandardOutput = true;
                a.StartInfo.UseShellExecute = false;
                a.Start();
                a.WaitForExit();
            }
```

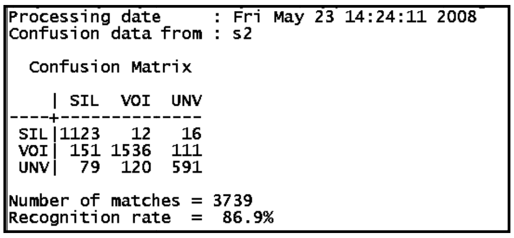
The above code snippet call the following SFS command for each file in the group to be imported into the SFS file:

```
anload -h recfile  sfsfile
```

After importing all recognition files into the associated SFS files, each SFS file will contain two annotation items. The reference annotation item and the recognized annotation item will be used by SFS to estimate the recognition rate. The results will be obtained by comparing the reference annotation item to the recognized annotation item for each 10(ms) of the spoken period. The following two SFS commands perform the annotation comparison. The output is a confusion matrix. The following lists the commands:

```
ancomp -r an.02 -t an.03 -f -m - AcPA0001.sfs > s1
Conmat s1
```

The above two commands are applied on all testing SFS files. The generated confusion matrix is like the one in figure 5.



```
Processing date     : Fri May 23 14:24:11 2008
Confusion data from : s2

  Confusion Matrix

    | SIL  VOI  UNV
----+--------------
 SIL|1123   12   16
 VOI| 151 1536  111
 UNV|  79  120  591

Number of matches = 3739
Recognition rate  =  86.9%
```

Figure 5: Confusion matrix for a certain recognition process.

The confusion matrix gives allot of information for the recognition process. Figure 5 gives such an example of the matrix. In this matrix we can notice that SIL sounds is recognized as SIL in 1123 matches out of (1123+12+16 = 1151). SIL is recognized as VOI for 12 times and as UNV for 16 times.

7. Results

The results of this research will be analyzed in this section. We have two sets of files. SET_A deals with Filter banks (VOC19) and SET_B deals with Best Tree 4 point Encoded features (BTE).

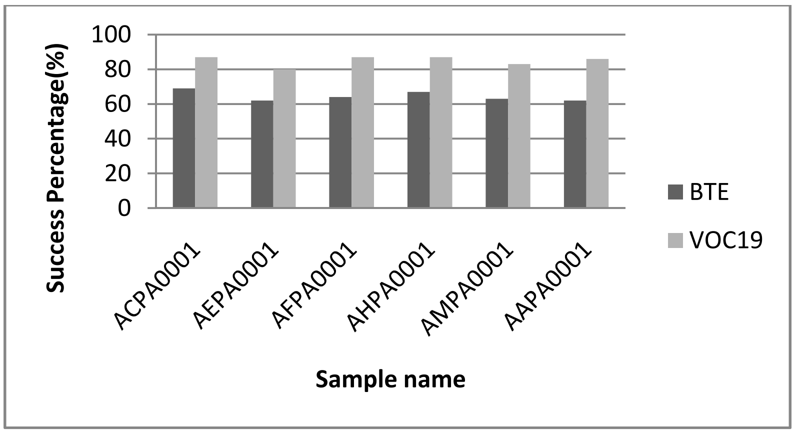


Figure 6: Comparison chart of overall system performance.

Figure 6 indicates the overall performance. As it is the first round in using BTE, Filter Banks (VOC19) features indicate a significant better performance than BTE. Many enhancements still may be added in the future to go around the drawback in the currently proposed BTE features. The detailed analysis of the results is introduced below to figure out the obtained results. As it will be shown below, the features failed in recognizing the UNV sounds while it makes comparable results in recognizing SIL and VOC. Figures 7, 8 and 9 provide the comparison results for the three classes under test on both features {BTE and VOC19}.

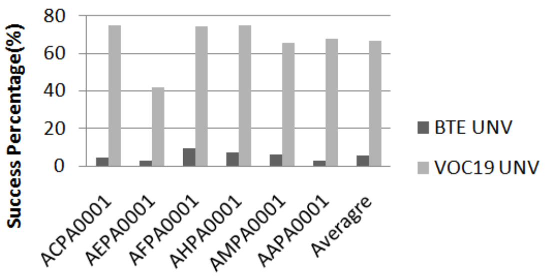


Figure 7: Comparison between BTE and Filter banks in recognizing (UNV) sound.

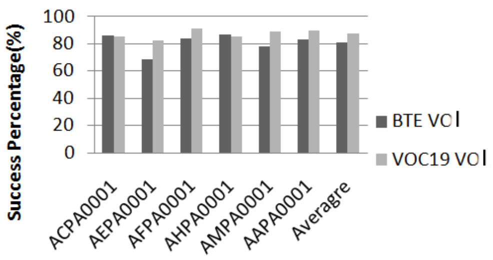


Figure 8: Comparison between BTE and Filter banks in recognizing (VOI) sound.

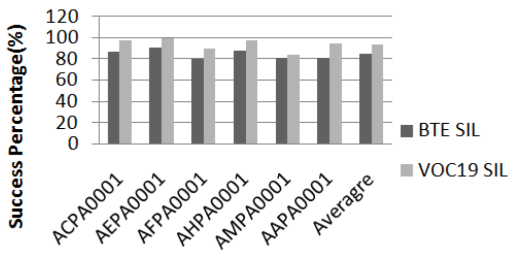


Figure 9: Comparison between BTE and Filter banks in recognizing (SIL).

8. Conclusions

This is a preliminary study to introduce BTE features. Many enhancements may be included in the future to minimize the confusion results being discussed in section 7.

9. References

[1] Amr M. Gody,"Wavelet Packets Best Tree 4-Points Encoded (BTE) Features", The 8th Conference on Language Engineering.2008, Cairo, Egypt.

[2] Mel scale, http://en.wikipedia.org/wiki/Mel_scale

[3] Giridharan, K. Smolenski, B.Y. Yantorno, R.E, "Statistical and model based approach to unvoiced speech detection", Intelligent Signal Processing and Communication Systems, 2004. ISPACS 2004. Proceedings of 2004 International Symposium, On page(s): 816 – 821

[4] University College London: http://www.phon.ucl.ac.uk/resource/sfs/howto/htk.htm.

[5] http://htk.eng.cam.ac.uk/