SPOKEN LANGUAGE ACCENT DETECTION

Probabilistic Accent Detection Using Hidden Markov Models

Tré Calhoun
La Vesha Parker
Andrew Vaslas
Nicolas Vera
Cornell University

CONTENTS

INTRODUCTION

Various accents pose a problem to automated speech recognition software. If accents can be more easily detected, different spoken language models can be applied to speech recognition software to make for a more correct interpretation of spoken language.

In this tutorial, we describe our exploration of Cambridge University's Hidden Markov Model Toolkit as a tool to use for spoken accent prediction. We explore the classification of the pronunciation of the word ”security” as spoken by native English and native Spanish speakers.

We provide additional data in `data_general/` for those who wish to explore additional accent classifications.

PART I

THE HIDDEN MARKOV MODEL TOOLKIT SOFTWARE SUITE

CHAPTER 1

INSTALLATION OF HTK SOFTWARE

1.1 General Installation information

The website for HTK can be found here. The HTK developers require that you register for a username and password through their site before downloading their software. After registering, visit the downloads page and download the HTK source code (available as a tarball). It is also useful to download the HTKBook as a PDF (available on the downloads page, below the software). If you do not wish to download the book, you can view the book online after registering.

1.2 Mac OS X

In order to install HTK for Mac OS X, you first need to make sure that you have Xcode developer tools and X11 installed.

What follows are the installation instructions taken *directly from the README in the root directory of the unziped `htk/` directory*, save a bit of formatting. We do not claim this work, and repeat it here only for convenience.

1.2.0.1 Compiling & Installing HTK under UNIX/Linux, OS X or Cygwin

After unpacking the sources, cd to the `htk/` directory.

There are now two ways to install HTK, the "traditional" and the "new". Up to now HTK has always installed its tools as they were built, and installed them to a directory such as "`bin.linux`" so that binaries for different architectures can be installed in a home directory say. If you want to install in this way, please add the option "`--enable-trad-htk`" when you run `configure`.

The "new" method installs by default into `/usr/local/bin` (equivalent to a configure option of "`--prefix=/usr/local`").

1. decide which of the above methods you wish to use
2. `cd` to htk, then run `./configure` (with appropriate options, run "`./configure --help`" if unsure). If you don't want to build the programs in HLMTools add the –disable-hlmtools option.
3. `make all`
4. `make install`

Running "`make install`" will install them. This step may need to be done as root, if you are not installing them in your home directory.

Notes for particular Unix variants:
Solaris: if "`make`" isn't installed you may need to add `/opt/sfw/bin` and `/usr/ccs/bin` to your path and run "`./configure MAKE=gmake`" with any other options you require. Then run "`gmake`" instead of "`make`", alternatively you can create a symbolic link called "`make`" somewhere it your path to `/opt/sfw/bin/gmake`

1.3 Windows

Once again, what follows are the installation instructions taken *directly from the README in the root directory of the unziped `htk/` directory*, save a bit of formatting. We do not claim this work, and repeat it here only for convenience.

1.3.0.2 Compiling & Installing HTK under Windows

Prerequisites:

- HTK has been verified to compile using Microsoft Visual Studio.
- For testing, you will require a Perl interpreter such as ActivePerl.

- You will need a tool such as 7-zip or winzip (commercial) for unpacking the HTK source code archive.
- It it is helpful if you have some familiarity with using the DOS command line interface, as you will need to interact with it in order to compile, install and run HTK.
- Ensure that your PATH contains:

  `C:\Program Files\Microsoft Visual Studio .NET 2003\Vc7\bin`

  Or if you are using older versions:

  `C:\Program Files\Microsoft Visual Studio\VC98\bin`

Compilation:

1. Unpack the HTK sources using 7-zip.
2. Open a DOS command window: Click Start, select Run type `cmd` at the prompt and click OK.
3. `cd` into the directory in which you unpacked the sources.
4. `cd` into the `htk/` directory. Type:
   ```
   cd htk
   ```
5. Create a directory for the library and tools. Type:
   ```
   mkdir bin.win32
   ```
6. Run `VCVARS32` (it should be in your path, see prerequisites above)
7. Build the HTK Library, which provides the common functionality used by the HTK Tools. Enter the following commands:
   ```
   cd HTKLib
   nmake /f htk_htklib_nt.mkf all
   cd ..
   ```
8. Build the HTK Tools
   ```
   cd HTKTools
   nmake /f htk_htktools_nt.mkf all
   cd ..
   cd HLMLib
   nmake /f htk_hlmlib_nt.mkf all
   cd ..
   cd HLMTools
   nmake /f htk_hlmtools_nt.mkf all
   cd ..
   ```

Installation:
The HTK tools have now been built and are in the `bin.win32` directory. You should add this directory to your `PATH`, so that you can run them easily from the command line in future.

CHAPTER 2

TRAINING AND TESTING CORPUS ACQUISITION

> Everyone has the right to life, liberty and security of person.
>
> —United Nations' Declaration of Human Rights [4]

2.1 The Online Speech/Corpora Archive and Analysis Resource (OSCAAR)

Northwestern University's Online Speech/Corpora Archive and Analysis Resource (OSCAAR) is a collection of speech recordings from speakers with different backgrounds, assembled from various datasets.

To request access to the data available through OSCAAR, you can submit a request for access to the OSCAAR collections. In our experience, requests are handled about 24 – 48 hours after being sent.

The dataset that we found most appropriate for our goal of accent detection and classification is the ALLSTAR dataset from the Speech and Communication Research Group at Northwestern University. The dataset is massive, and we found that a subset of samples fit our needs well.

Part of the dataset features recording of talkers from different backgrounds saying 20 sentences pulled from the Declaration of Human Rights in English. For our proof of concept, we used a subset of that portion of speakers reading Article 3 from the DHR: ”Everyone has the right to life liberty and security of person.” That subset of the data featured talkers with the following native tongues:

- Brazilian Portuguese
- English
- French
- German
- Hebrew
- Hindi
- Japanese
- Korean
- Mandarin Chinese
- Persian (Farsi)
- Russian
- Spanish
- Turkish
- Vietnamese

We use the Spanish and English samples, but make all of them available in `data_general/`.

CHAPTER 3

TRAINING CORPUS WITH HTK

3.1 Record or Input Sound Files

It is possible to record one's own audio for this classification using HSLab. For those like us who already have a dataset and will not use the recording features of HSLab, you can use a configuration file to change the anticipated input format to the program.

Having acquired the sound clips from OSCAAR, we needed to select two distinct native tongues. Native English and native Spanish speakers were selected since these subsets had what we felt was a sufficient amount of data for the purposes of this project; some of the others only had a few samples and having more data is conducive to the training of the HMM.

About $\frac{1}{4}^{th}$ of our data was set aside for testing. In `data/train/` and `data/test/`, the `.wav` audio clips themselves are stored in `wav/` and their respective `.mfcc` and `.lab` files were stored in `mfcc/` and `lab/`. The free audio editor Audacity was used to crop the `.wav` files so that only the word "security" could be heard. Audacity allowed us to generate silent audio surrounding the cropped clips for easier labelling.

3.2 Labeling the Sound Files

HSLab allowed us to label the boundaries between words and the silence around them in each `.wav` file. With a beginning silence, the spoken word, and the ending silence labeled, a `.lab` file for each clip was created. The `.lab` files are plain text files marking the start and end sample times for each of these labeled sections:

```
data/train/lab/english_f1_security.lab
20408 4239909 sil
4342857 9941497 security_english
9982766 13996825 sil
```

CHAPTER 4

CODING THE DATA

4.1 Mel Frequency Cepstral Coefficients

Here we describe what a MFCC is and its usefulness to us.

4.2 Obtaining `.mfcc` Files

The .wav files themselves cannot be analyzed using HTK, so we used HCopy to convert the original `.wav` files into `.mfcc` files. The `.mfcc files`, which each contain a set of vector representations of the sound signal, can be analyzed. Each 25ms segment is represented by a vector of acoustical coefficients, which provides a description of that segment's spectral properties. HCopy requires a configuration file to specify its parameters:

4.2.1 Configuration File

```
#analysis.conf
SOURCEFORMAT = WAV    # Gives the format of the speech files
TARGETKIND = MFCC_0_D_A    # Identifier of the coefficients to use
```

```
# Unit = 0.1 micro-second :
WINDOWSIZE = 250000.0    # = 25 ms = length of a time frame
TARGETRATE = 100000.0    # = 10 ms = frame periodicity

NUMCEPS = 12     # Number of MFCC coeffs (here from c1 to c12)
USEHAMMING = T    # Use of Hamming function for windowing frames
PREEMCOEF = 0.97     # Pre-emphasis coefficient
NUMCHANS = 26     # Number of filterbank channels
CEPLIFTER = 22     # Length of cepstral liftering
```

4.2.2 The Creation of `targetlist.txt`

The file `trainlist.txt` gives the name and directory of each waveform file and their respective target coefficient files:

```
data/train/wav/english_f1_security.wav data/train/mfcc/english_f1_security.mfcc
data/train/wav/english_f2_security.wav data/train/mfcc/english_f2_security.mfcc
```

⋮

```
(and so on)
```

4.3 Command Line Actions

Use the following command to execute this conversion:

```
HCopy -A -D -C analysis.conf -S trainlist.txt
```

CHAPTER 5

SETTING PARAMETERS FOR THE HIDDEN MARKOV MODEL

5.1 What is a Hidden Markov Model?

A hidden Markov model (HMM) is a type of Markov model, which means future states depend only on the current state. Classically, the states in this model are classified as one of two types: observed and hidden. This model states that the observed states are determined by the underlying hidden states; thus, the observations are inputs to the problem and the hidden states need to be discovered. There are two key types of probabilities that connect the states: emission probabilities, the probability of an observation state given a hidden state; and transition probabilities, the probability of a hidden state given a previous hidden state (the Markov property). Given a number of hidden states and a set of observations, we can learn the probabilities to maximize the collective probability of each observation. Given the states, the transition and emission probabilities, and a sound, we can use a HMM to determine the probability of the sound under the model.

5.2 HMMs and Accent Detection

5.2.1 Overview

A hidden Markov model is a good model to use for accent detection because a discretized sound directly maps to the observation states as inputs to the model. These observed sounds segments are dependent on the speaker's accent, which means there must be underlying hidden states that model the likelihood of the sound segment produced given the accent (the emission probability). It is reasonable to believe that transition probabilities exist between the hidden states, as accents may dictate that certain sounds should have a high or low likelihood of following another sound. We use HMMs by defining an HMM for each of the acoustical events (English, Spanish, and silence). Then, given a sound and its correct event type, we want to train each HMM (i.e. determine the transition and emission probabilities) so that the corresponding HMM with the same event type returns a probability that is much higher than the the probabilities that the others return. Finally, given a sound, we can predict its acoustical event type using the one-vs-all methodology by calculating which HMM returns the highest probability. More details about the process are below.

5.2.2 HMM Definition

To implement an HMM, we must first define its structure. We define each HMM as a continuous density HMM with n states in total, 2 of which are non-emitting (HTK reserves the first and last states due to its internal implementation). Due to the fact that each observation (a 25ms segment) is represented by a vector of acoustical coefficients, our emission probability must actually be a vector of sub-emission probabilities corresponding to each acoustical coefficient. As each acoustical coefficient is a floating-point number, we describe the probability of the coefficient by using a Gaussian distribution. Gaussian distributions are defined by a mean and variance; thus, we can simply describe the overall emission of each hidden state by a vector of means and a vector of variances. (The cardinality of the mean and variance vectors is equal to the number of acoustical coefficients.) We also define a $n \times n$ transition matrix that defines the transition probabilities between hidden states.

We define "prototype" HMM description files for each of the acoustical events that contains this information; each file is equivalent, save for different names. These files are prototypes, as they describe the structure of the HMM, but the values (mean and variance vectors and transition probabilities) are relatively arbitrary, and will be corrected during initialization and training (described below). Thus, we set each mean as a vector of zeros, each variance as a vector of ones, and the transition probabilities are established so that the sum of transitioning from state i to any state is equal to 1 (except for the final, nth state, which is accepting, so all of its transition probabilities equal 0). Note that setting any index of this transition matrix to 0 means that it will always be 0, even after initialization and training. We use $n = 6$ states, and mean and variance vector sizes of 39, as there are 39 MFCC acoustical coefficients.

```
~o <VecSize> 39 <MFCC_0_D_A>
~h "security_english"
<BeginHMM>
      <NumStates> 6
      <State> 2
         <Mean> 39
           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 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
         <Variance> 39
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <State> 3
         <Mean> 39
           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 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
         <Variance> 39
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <State> 4
         <Mean> 39
           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 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
         <Variance> 39
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <State> 5
         <Mean> 39
           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 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
         <Variance> 39
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
           1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      <TransP> 6
        0.0 0.5 0.5 0.0 0.0 0.0
        0.0 0.4 0.3 0.3 0.0 0.0
        0.0 0.0 0.4 0.3 0.3 0.0
        0.0 0.0 0.0 0.4 0.3 0.3
        0.0 0.0 0.0 0.0 0.5 0.5
        0.0 0.0 0.0 0.0 0.0 0.0
<EndHMM>
```

The text above shows our prototype HMM definition for the English accent. These prototypes are saved as model/proto/hmm_security_english, model/proto/hmm_security_spanish, and model/proto/hmm_sil for the corresponding sound type in a model/proto/ directory.

5.2.3 Training

Training is the process of estimating the parameters of the HMMs by using labelled sound examples. To start, we initialize each HMM with the HTK tool HInit, which time-aligns the training data with a Viterbi algorithm. Doing this estimates initial parameters (mean and variance vectors and transition probabilities) based of off the prototype HMM description file for each initial HMM. Then we train the models by using the HTK tool HRest on each HMM until convergence. HRest uses Baum-Welch parameter re-estimation to perform one re-estimation iteration on an input HMM, producing a new HMM. Starting from the initialized HMMs, we iteratively use HRest until its change measure does not decrease (i.e. until it converges). We performed one iteration on the Silence HMM and three iterations on the English and Spanish HMMs.

This completes the training of the HMMs; we now have models such that if we input a single acoustical event (a sound of silence, a sound of ”security” in an English accent, or a sound of ”security” in Spanish accent), we can predict which event type it is with some accuracy. More detail about HInit and HRest can be found in the HTKBook.

5.3 Command Line Actions

The following command initializes an HMM with HInit:

```
HInit -A -D -T 1 -S trainlist.txt -M model/hmm0
      -H model/proto/hmmfile -l label -L label_dir nameofhmm
```

Here:

- hmmfile is contained in model/proto/hmm_security_english, model/proto/hmm_security_spanish or model/proto/hmm_sil
- label is ”english”, ”spanish”, or ”sil”
- label_dir is in data/train/lab/
- nameofhmm is ”english”, ”spanish”, or ”sil”

This must be repeated for each model: English, Spanish, and sil.

The following command performs one re-estimation iteration with HRest:

```
HRest -A -D -T 1 -S trainlist.txt -M model/hmmi -H vFloors
      -H model/hmmi-1/hmmfile -l label -L label_dir nameofhmm
```

Here:

- model/hmmi refers to the output directory, where i indicates the index of the current iteration ($i = (1, 2, 3, \ldots)$)

- `hmmfile` is contained in `model/proto/hmm_security_english, model/proto/hmm_security_spanish` or `model/proto/hmm_sil`
- `model/hmm`$i-1$ indicates the index of the last iteration $(0, 1, 2, \ldots)$
- `label` is ”english”, ”spanish”, or ”sil”
- `label_dir` is `data/train/lab/`
- `nameofhmm` is ”english”, ”spanish”, or ”sil”

This procedure has to be repeated several times for each HMM to train. We need to train each hmm until it converges. An indicator of convergence is when the number of iterations each step no longer decreases. The number of iterations per step is reported on the line that reads: ”Estimation converged at iteration <number>.”

CHAPTER 6

DEFINING YOUR TASK

6.1 Define Your Grammar

Once you have created HMMs for each of the accents you'll be including in your test sample, you're ready to define the task. The first step in defining the task is creating a grammar, which contains the syntactic structure of examples to be tested. In our case, the grammar is quite simple. It consists of a start silence, the word "security", and an end silence. The word "security" can be in spoken with either a Spanish or American English accent. Thus, we define the grammar file as follows:

```
$WORD = ENGLISH | SPANISH;
( { START_SIL } [ $WORD ] { END_SIL } )
```

Essentially, this means that we have a variable called WORD that can take the value SPANISH or ENGLISH. Additionally, the brackets, {}, indicate one or more occurrences of that which they enclose, and the other brackets, [], indicate zero or one occurrence of their inner contents. Given these definitions, the syntactic structure of our sample, as indicated by the second line in our grammar file, is one or more repetitions of START_SIL, zero or one occurrence of either SPANISH or ENGLISH, and one or more repetitions of END_SIL.

6.2 Define Your Dictionary

Now that we have defined our task grammar, we must connect the grammar to the HMMs we developed in the previous chapter. In other words, our system must be able to associate each variable (SPANISH, ENGLISH, START_SIL, END_SIL) with an HMM. To do this, we create another simple file called the task dictionary as follows:

```
YES [yes] yes
NO [no] no
START_SIL [sil] sil
END_SIL [sil] sil
```

Here, the elements in the leftmost column obviously correspond to the task grammar's variables. The elements in the rightmost column indicate the HMMs to which each of the variables corresponds. The elements in the middle column specify the symbols that will be output in the final recognition step. This middle column is optional; by default, the recognizer will output symbols corresponding to the task grammar variables' names. IMPORTANT NOTE: Do not forget the new line at the end of the dictionary file. Failure to include it will result in the last entry (in this case, END_SIL) being ignored.

6.3 Generating the Network

Finally, you are ready to create the network, which will, in essence, serve as a finite state machine (FSM) through which you can run additional samples to generate labels. To do this, we use HParse to compile the grammar (`gram.txt`) into our network. We use the following command to write our network to file `net.slf`:

```
HParse -A -D -T 1 gram.txt net.slf
```

To test that the network is valid and ready for testing, use HTK tool HSGen to generate random sentences that should conform to the syntactic regulations as specified in the grammar. The following command can be used, assuming the dictionary is defined in `dict.txt`:

```
HSGen -A -D -n 10 -s net.slf dict.txt
```

This should output 10 (as specified by the argument passed to -n) sentences. Check these to ensure they are in accordance with your grammar rules.

CHAPTER 7

RECOGNITION

7.1 Procedure

Once you've generated a valid network that includes your trained HMMs, you are ready to classify new sound samples into accent bins. If you did not set aside MFCC samples for testing, you must once again transform an input signal file into MFCCs using HCopy (as you did with the training data). Then, we use an implementation of the Viterbi Algorithm to pass the input through the network and generate a label.

7.2 Command Line Actions

The tool used to run the new test sample (MFCC file) through the network is HVite. Use the following command to test file `input.mfcc`:

```
HVite -A -D -T 1 -H hmmsdef.mmf -i reco.mlf
      -w net.slf dict.txt hmmlist.txt input.mfcc
```

Here, `input.mfcc` is the input data we'd like to label, `hmmlist.txt` lists the names of the models to use (separated by new line characters), `dict.txt` is the task dictionary, `net.slf` is the task network, `reco.mlf` is the output recognition file and `hmmsdef.mmf` is a single file containing a concatenation of the HMMs to be used. If, instead of creating this file, you'd prefer to list each HMM separately, you can do so by replacing `hmmsdef.mmf` with `-H hmm_security_english -H hmm_security_spanish ...` The output file (`reco.mlf`) lists the word "hypotheses" made by the network for each recognized segment of the input file, along with the start points and end points of each.

PART II

ERROR HANDLING, SOFTWARE USED AND RESOURCES

APPENDIX A

ERROR HANDLING

Errors that crop up when using HTK can seem confusing initially, but they can be debugged with a bit of critical thinking and some hints from outside sources. We found the following sources helpful when debugging:

- Ohio State University: Understanding HTK Error Messages
- Columbia University: Summary of Errors by Tool and Module

APPENDIX B

SOFTWARE USED

We have provided information about our use of various software (particularly the various toolkits available through the HTK software) throughout this guide, so we will merely provide a summary of what

1. Audacity: Useful for processing audio files. Used to splice .wav files to extract specific words from .wav files. Also useful for generating segments of silence to make differentiating between SIL labels and spoken words easier.

2. HTK: Maybe even list each of the things we used under HTK & why, i.e. HSLab for labeling, HParse for whatever

   (a) **HSLab**
   Provides a graphical user interface for the labeling of sound files. Accepts waveform files (i.e. .sig files recorded directly in HSLab and .wav files that users can pre-record). The default expected filetype is `.sig`, so if you plan to use a different file type, be sure to include a configuration file (i.e. our `analysis.conf`) that specifies the SOURCE FORMAT.

   Sample command line invocation:

   ```
   HSLab -C <config_file.conf> <sound_file.ext>
   ```

(b) **HCopy**
The primary use of HCopy is to copy and manipulate speech files. Another use for HCopy (and our particular use here) is to convert waveform data to Mel Frequency Cepstral Coefficients. HCopy accepts pairs of source specifications for `.lab` files and destination specifications for the `.mfcc` files that HCopy will generate for each of the `.lab` files.

Sample command line invocation:

```
HCopy -C <config_file.conf>  -S testlist.txt
```

(c) **Hinit**
Used to initialize each of the hidden Markov models that will be trained using HRest to model each of the accents (english, spanish) and silence (soil).

Sample command line invocation:

```
HInit -A -T 1 -S hinit_trainlist.txt -M model/hmm0
      -H model/proto/hmm_security_english -l security_english
      -L data/train/lab security_english
```

(d) **HRest**
Trains each HMM.

Sample command line invocation:

```
HRest -A -T 1 -S hinit_trainlist.txt -M model/hmm1
      -H model/hmm0/hmm_security_english -l security_english
      -L data/train/lab security_english
```

(e) **HParse**
Compiles the task grammar into a task network.

Sample command line invocation:

```
HParse -A -T 1   def/gram.txt   net.slf
```

(f) **HVite**
Allows you to test your trained HMMs and grammar on a new sample.

Sample command line invocation:

```
HVite -A -D -T 1 -H hmmsdef.mmf -i reco.mlf
      -w net.slf dict.txt hmmlist.txt input.mfcc
```

APPENDIX C

REFERENCES

REFERENCES

[1] "Hidden Markov Model".

[2] 'HTK Basic Tutorial'.

[3] 'HCopy Config File'.

[4] UN General Assembly, *Universal Declaration of Human Rights*, 10 December 1948, 217 A (III), available at: http://www.refworld.org/docid/3ae6b3712c.html