

Problem Set 6 and Lab 3

Problem Set 6

1. Assuming an N state continuous observation hidden Markov model Φ with M Gaussian mixtures, describe the probability (e.g. $\Pr(\dots | \dots)$) that is specified by the formula $\sum_{k=1}^M \zeta_t(j, k)$ for any $1 \leq t \leq T, 1 \leq j \leq N$ (20 points).
2. The htk-resources directory (see lab below for location of files) contains a Matlab file “seven.m” which contains the parameters of a 4 mixture whole word model for the word “seven.” You can read it to see the structure. Write a Matlab subroutine that returns the log probability of the best path and the path itself when given a model in format stored in seven.m and a matrix of feature vectors.

Compute the log best-path probability and associated path for the given model seven using the cepstral feature vectors for the first instance of female test speaker lr saying the number 7 ‘.../tidigits/mfc/test/woman/lr/7b_01.mfc’. You may reuse any code from previous assignments or solutions (40 points).

HINT: Use a log probability implementation

3. Assume that a speech recognition system uses 3 state models for triphones. As an example, the word elephant [ɛləfənt] could be written as six triphones (20 points):

*-ɛ + l, l-ə + f, ə-f + ə, f-ə + n, ə-n + t, and n-t + *.

Using the dictionary presented in class when discussing lexical baseforms, write the trigram models that would be used for the words seven and zero. Suppose we wanted to tie the center state of triphone models. Give one example of two triphone models whose center state could be tied. Be sure to indicate from which word each triphone originates. What are we trying to achieve by tying a state?

4. Why are lexical baseforms for proper names typically modeled by machine learning techniques as opposed to rules (20 points)?

Laboratory 3

In this assignment, you will construct a continuous speech digit recognizer. You will begin by constructing a speaker independent model using the TIDIGITS corpus, a collection of studio quality digit speech collected by Texas Instruments. A copy of a

subset of this corpus has been placed in both the Windows lab and rohan. An overview of the corpus can be found in corpora/tidigits/readme.doc (a text file that can be read with any editor, not a Microsoft Word file). The following are subdirectories of TIDIGITS which will be of interest to you:

| | |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| doc | Tables describing the speakers, their dialects, and how the data was originally stored & labeled. The readme.doc file in the main directory contains more recent information about the file naming conventions. |
| train | Training source data for male and female speakers, child speakers have been omitted. |
| test | Test data for male and female speakers, child speakers have been omitted. |
| mfc | The first 12 Mel filtered cepstral coefficients which have been extracted from the waveforms in the sphere directories. |
| htk-resources | A set of files which will be useful for performing speech recognition with HTK |

Before you begin, you should make sure that the following directories are included in your shell's PATH variable:

| | |
|---------------------------------|------------------------------|
| rohan.sdsu.edu | Windows lab |
| ~mroch/htk/bin/bin.sun4_solaris | \\cathouse\htk-3.4\bin.win32 |
| ~mroch/bin-htk | \\cathouse.sdsu.edu\bin-htk |

When you listen to the train/test data in the lab, remember that the data is stored in the SPHERE file format. You must install the SPHERE plugin in order to read this data type in Wavesurfer. This can be done by selecting the Help→About Plug-Ins menu, and choosing to install the plugin for SPHERE.

It is suggested that you may want to modify your shell startup files so that your PATH is adjusted automatically each time you log in. See the class FAQ for directions on how to do this.

Background

While HTK provides a powerful way to construct hidden Markov model based recognizers, it is quite tedious to use. Each step requires the execution of a separate command line program and students in previous classes have found the experience of using HTK less than rewarding. We will be using a Python script which automates much of the “drudge work” associated with HTK. You will be expected to refer frequently to *The HTK Book*, the directions here are intended as a supplement, not a replacement.

Given the limited vocabulary of our recognizer, it would be feasible to create whole word models, but we will create isolated phoneme models so that you gain experience creating subword models. It is highly recommended that you read through this entire assignment and reference the HTK book before starting the exercise. Make sure that you have a good concept of the “big picture” of what you are doing rather than just executing commands.

In this lab, going through the steps and understanding what you are having HTK do is much harder than the actual steps themselves. You do not need to write any code to complete the experiments, but you will need to author some configuration files. Your lab report should describe the theoretical steps and demonstrate a clear understanding of why each step is necessary.

Getting Started

Our task is to recognize the digits zero through nine and the word “oh”. To perform our experiments, we will be needing several things¹:

1. A set of lexical baseforms describing how our words are represented by phonemes. A subset of the [CMU Pronouncing Dictionary](#) has been stored in the file *dictionary*. A list of all of the phonemes used and the symbol *sil* for silence are in file *digit_monophones*. Note that the digit ONE can be pronounced in two ways which are differentiated by ONE and ONE(2). When writing your grammar, you do not need to make the distinction. Writing ONE will generate HMM networks for both forms.
2. A list of all of the feature files used for training and testing. These are stored in HTK script files *train_m_01.scp/train_f_01.scp* and *test_m_01.scp/test_f_01.scp*. Script files are described in section 4.7 of the HTK Book, but are simply files containing lists of files on which HTK will operate.
3. As HMMs are supervised classifiers, labels for the training data are required. In order to evaluate how well the system works, labels are also required for the test data. HTK permits two ways of specifying the labels:
 1. For each feature file, HTK expects a second file ending in the extension *.lab* which contains the labeled data.
 2. Alternatively, a format is permitted where the contents of all the label files are grouped into a single file called a master label file.

We will be using the second method. Files *train_mw.mlf* and *test_mw.mlf* are master label files for the male and female TIDIGITS data.

4. Configuration files describing feature extraction and the resulting feature set. The feature extraction configuration files are *mic_01* and *mic_02* with varying extensions. The extensions indicate what the source material is: *.sph* and *.wav* extract features from sphere and wav files respectively, and *.htk* can be used to extract features directly from live microphone input. Regardless of the source material, they are all designed for microphone speech and have Mel filters spanning 75 to 7,500 Hz. The 01 configuration uses the first 12 MFCC, and the

¹ Some of the files needed for HTK have been placed in the *htk-resources* directory for you.

02 configuration adds first and second derivatives. Read about configuration files in the HTK Book.

The dictionary

The file *dictionary* has already been created for you, but you should read the corresponding section of the HTK manual to understand how dictionaries are created.

The grammar

As this is a small vocabulary task, we will not be creating an N-gram language model, but will instead create a small context free grammar. Step 1 of the HTK tutorial in the HTK Book tells you how to create a grammar for a phone dialing task. Read these instructions then write a grammar (save it as file *grammar*) which recognizes one to seven digits with the optional symbols SENT-START and SENT-END before and after the digits. If you consult the dictionary file, you will see that both SENT-START and SENT-END are mapped to the silence model, meaning that you can optionally have some silence before and after the digits. Be careful as HTK is picky about file formats. Last lines that do not end with a new line and blank lines are likely to result in cryptic error messages.

Use HParse to compile the grammar to HTK's internal representation which is called a word net (see the HTK tutorial). Use "wdnet" as the output file name. You can test your grammar to see if it works correctly by generating a set of random sentences with the following command:

HSGen wdnet dictionary

which will generate 100 sentences (the number can be set with the `-n` option). The following is an example of a few sentences generated by HSGen:

```
ZERO
THREE EIGHT ZERO NINE FIVE ONE
OH THREE SEVEN ONE
OH SIX
THREE
FIVE FOUR
EIGHT EIGHT SEVEN
NINE SIX OH SEVEN TWO
```

Note that the SENT-START and SENT-END symbols are never generated. This is because the dictionary file specifies that these symbols should never be output.

Transcription files and Feature Extraction

The master label files contain word level labels for the each of the feature files. As an example, the feature file *train/woman/bh/93623a_01.mfc* has a corresponding transcription in *train_mw.mlf* as follows:

```
"/train/woman/bh/93623a_*.lab"  
NINE  
THREE  
SIX  
TWO  
THREE  
.
```

This would be sufficient to train a whole word model, but it must be converted to a phonetic representation to train subword models. This can be done with the HTK Label Editor (HLEd). For our purposes, we will need to provide HLEd with a command file which gives instructions on how the master label files should be transformed. We will put the following two commands in a file called `mkphones0.led`:

```
EX  
IS sil sil
```

Read about HLEd in the HTK Tool reference section of the HTK Book then execute the following command which will create a master label file with words expanded to phonemes²: In addition to the tutorial section, you can see details on any HTK command in section 4. The EX *expands* words to phonemes and the IS *inserts* the first label before each transcription and the second after the transcription. Here, we are adding silence to the beginning and end of each transcription. Think about why we might want to do this. The label editor can be executed as follows:

```
HLEd -d dictionary -i phones0.mlf mkphones0.led train_mw.mlf
```

Compare `train_w.mlf` and `phones0.mlf` to see the results of your actions.

Training features have already been extracted for you and are in the `mfc` directory. **Do not copy the TIDIGITS feature data to your home directory, it is likely to exceed your disk quota.**

Flat Start Training

A set of monophone single mixture HMMs will now be created using flat start training. The HTK tutorial asks that we do the following steps:

² We will assume that you copied `dictionary` and `train_mw.mlf` to your current working directory. If you did not, you will need to provide the complete path to these files. This will be true for all of the examples throughout this tutorial. Note that the training master label file contains labels for both genders, HTK simply takes what it needs.

1. Create a single mixture prototype model by hand with 3 states. Note that HTK uses the concept of “non emitting” states to provide entry and exit to construct networks of HMMs, so the first and last state will not have any observations associated with them. This is an alternative to the null transitions that we studied in class. Consequently, the model will have 5 states, but only 3 of them will be associated with observations.
2. Determine the mean and variance of all of the training data, regardless of class. This is the so called “grand mean and variance.” It can be estimated with the tool HCompV.
3. Manually create copies of the prototype for each phoneme to be recognized.
4. Execute iterations of the EM algorithm until a stopping criterion is met. In HTK, each iteration is performed by the command HERest.

You should read section 3.2.1 and make sure that you understand what it is doing. One of the concepts that students frequently have problems with is the HTK’s concept of macros. Each HMM is represented in an ASCII file³. Macros start with a ~ and have a character which indicates the macro type. In this exercise, there are three types of macros which will be used:

- ~o – The global options macro. HTK expects this macro at the beginning of a file containing models. The macro specifies things about the feature set. Sample information: What types of features are being used (e.g. MFCC)? What is the dimensionality of the feature vectors? Are there any derived features (e.g. _D)?
- ~v – Variance floor macro. Used for setting the variance floors.
- ~h – HMM macro. Each HMM macro has a string following it which is the model name followed by a definition of the HMM characteristics as described in the HTK book.

These macros can be stored in one or more files which the HTK tools can be directed to load by using the `-H` option. While you will need to use this option during the testing phase, I have written a small program in Python called `train_hmm.py` which automates all of these steps.

The `train_hmm.py` Python program is located in the `bin-htk` directory. It automates many of the steps from the HMM tutorial for training. You should copy `bin-htk` and its subdirectories to your home directory/folder (on UNIX/linux: “`cp -rp ~mroch/bin-htk ~`”). If you prefer not to copy the files, you can set the environment variable `BINHTK` to point to `bin-htk`. Remember to set your `PATH` so that the `bin-htk` is included (see the course FAQ).

Create a directory where the trained models will be stored, perhaps: `mic_01/mix01` for microphone configuration 1, single mixture models. Then type on a single line⁴:

³ The `-B` option allows users to store the representation in binary format, which is appropriate when creating large model sets such as triphones.

⁴ In this example and all others, we are assuming that we are training for females. If you are male, use the male training and test data. Remember that the MLF files contain information for both genders and it is only the script files that are gender dependent.

```
train_hmm.py verbose=true global_opts="-T 1" isolated=0
classes=digit_monophones states=5 mixtures=1
mlf=phones0.mlf train=train_w_01.scp config=mic_01.mfc
outdir=mic_01/mix01
```

Invoking this command will create single mixture monophone HMMs. Note that all options are passed as keyword value pairs and order is not important. The options we used in this example are:

`verbose` – If set to true, will show each HTK command and its results as it is executed. You will probably want to set this to true for the first model or two you train and then let it default to false once you have an appreciation for the HTK commands that `train_hmm` is executing for you.

`isolated` – Indicates whether or not to build isolated word models. Since your speech is continuous this should always be set to 0.

`global_opts` – A list of options that are passed to all HTK tools. “-T 1” indicates that the trace level should be set to 1 which provides basic information about HTK’s progress.

`classes` – A text file containing a list of the names of the HMM models that HTK will be using. The file `digit_monophones` contains a list of all the phonemes that we are using.

`states` – The total number of states. Remember that 2 of these states are non-emitting, so setting states to 5 is equivalent to the 3 state models that we have often used in class.

`mixtures` – How many Gaussians are used?

`mlf` – The master label file which contains the supervisory information for all of the training data. Note that since we are using subword models, we are using the `phones0.mlf` file that you created earlier as opposed to the `train_mw.mlf` file.

`train` – A file containing a list of files which comprise the training data. Depending upon the specified configuration, the files named in the list can contain audio data or features.

`config` – This is the name of the HTK configuration file. It is typical for researchers to perform feature extraction once and then specify via the configuration file that the training list contains feature files. This way, feature extraction only needs to be done once. Compare the configuration file `mic_01.mfc` which is designed to work with feature files versus `mic_01.sph` and `mic_01.wav` which are designed to extract features and save them to a file.

`outdir` – This specifies the directory where the completed models are to be stored. By default, the file `model_defs.mmf` is stored in the output directory.

The `train_hmm.py` Python program begins by creating a prototype file and invoking `HCompV` to set the means and variances of the Gaussians associated with each state all to the same value. When you run the command, you will notice that the command prints the following statement:

```
Temporary File Root = /tmp/model...
```

which tells you which directory will be used to store all of the temporary files. Usually, the `train_hmm.py` program is configured to erase the temporary files once it is done, but that feature is disabled in your version so that you can examine what HTK did. Each subdirectory of the temporary root contains one version of the models. The prototype directory contains the prototype. Models are first created with a single mixture and stored in `m0001/model_defs.mmf`. The results of each iteration of the EM algorithm is then stored in `m0001-EMNN/model_defs.mmf` where NN is the NNth iteration. When more than one mixture is specified, the mixtures are split after a few iterations and new models are created in `m0002`. This process continues until the desired number of mixtures has been reached. As the temporary directories are not being deleted, it may be helpful to look at several iterations of a specific model to see how one of the parameters (e.g. the mean) converges. You could even plot it and include it in your report.

Testing with TIDIGITS

Once your model has been trained, you are ready to test. You will need to use two HTK tools directly for testing: `HVite` and `HResults`.

`HVite` performs the Viterbi decoding. It can be invoked as follows:

```
HVite -H DIR_CONTAINING_MODELS/model_defs.mmf -C
      mic_01.mfc -p 0 -I test_mw.mlf -i results.mlf -S
      test_w_01.scp -w wdnnet dictionary digit_monophones
```

This will read in your previously created models and test the specified files (`test_w_01.scp`) using a set of known transcriptions (`test_mw.mlf`). The recognized transcriptions are stored in `results.mlf`. Further details about this command can be found in *The HTK Book*, both in the reference section and section 3.4.

Executing `HVite` does not report the accuracy of the recognizer. To do this, you must execute another HTK command: `HResults`.

```
HResults -T 1 -p -I test_mw.mlf dictionary results.mlf
```

This will produce a report detailing how well the system worked. Here is a sample report from a 1 mixture recognizer on one of your datasets with an insertion penalty of -20.

```
===== HTK Results Analysis =====
Date: Tue Apr 24 14:05:11 2007
Ref : test_mw.mlf
Rec : results.mlf
----- Overall Results -----
```

SENT: %Correct=80.41 [H=3529, S=860, N=4389]

WORD: %Corr=95.41, Acc=92.39 [H=13762, D=156, S=506, I=435, N=14424]

| ----- Confusion Matrix ----- | | | | | | | | | | | | |
|------------------------------|------|------|------|------|------|------|------|------|------|------|------|----------------|
| | E | F | F | N | O | O | S | S | T | T | Z | |
| | I | I | O | I | H | N | E | I | H | W | E | |
| | G | V | U | N | | E | V | X | R | O | R | |
| | H | E | R | E | | | E | | E | | O | |
| | T | | | | | | N | | E | | | Del [%c / %e] |
| EIGHT | 1229 | 1 | 0 | 5 | 1 | 1 | 2 | 1 | 21 | 19 | 0 | 40 [96.0/0.4] |
| FIVE | 1 | 1248 | 15 | 27 | 1 | 5 | 0 | 1 | 8 | 0 | 0 | 4 [95.6/0.4] |
| FOUR | 0 | 8 | 1264 | 0 | 16 | 24 | 0 | 0 | 0 | 1 | 0 | 4 [96.3/0.3] |
| NINE | 2 | 4 | 1 | 1192 | 4 | 55 | 5 | 0 | 8 | 1 | 4 | 21 [93.4/0.6] |
| OH | 5 | 28 | 64 | 1 | 1066 | 24 | 7 | 0 | 0 | 6 | 3 | 66 [88.5/1.0] |
| ONE | 2 | 6 | 5 | 27 | 9 | 1237 | 3 | 0 | 0 | 1 | 0 | 7 [95.9/0.4] |
| SEVE | 2 | 1 | 0 | 1 | 1 | 3 | 1324 | 0 | 1 | 1 | 1 | 1 [99.2/0.1] |
| SIX | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 1319 | 0 | 8 | 0 | 0 [99.2/0.1] |
| THRE | 3 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1295 | 10 | 0 | 3 [98.9/0.1] |
| TWO | 10 | 0 | 0 | 1 | 0 | 0 | 7 | 6 | 0 | 1280 | 4 | 9 [97.9/0.2] |
| ZERO | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 1 | 1 | 2 | 1308 | 1 [99.4/0.1] |
| Ins | 105 | 26 | 13 | 16 | 132 | 64 | 0 | 3 | 1 | 73 | 2 | |

The interpretation of the results is explained briefly in section 3.4.1 and in detail in section 13.4 of *The HTK Book*. In the confusion matrix, the rows are the actual classes and the columns are the assigned labels. Thus the first row shows that eight was classified as an eight 1,211 times, once as a five, etc. The Del column indicates how many times this number was simply omitted. The %c indicates what percentage of the time the word was classified correctly. For EIGHT, $1229/1280=96.0\%$. The %e is a measurement of how much of the substitution error is caused by this symbol. For EIGHT, there were 51 misclassifications (the sum of the off diagonal components of EIGHT's row) amongst the $N=14,424$ words that needed to be classified: $51/N=.4\%$.

Train and test single mixture models using the mic_01.mfc configuration file. Next, test the 16 mixture models which I have trained for you and put in the htk-resources directory: digits_mix16_w_mic01.mmf or digits_mix16_m_mic01.mmf. There is a tendency for HMMs to favor shorter words over longer ones. The smaller number of feature vectors tends to result in a higher likelihood, so a common compensation technique is to add a penalty to the log likelihood each time a word is completed. Penalty values are task dependent, but try a few values between -5 and -40. The penalty value is specified with the -p option of HVite. Note the difference in the sentence error rates and the number of insertions and deletions (I and D) and plot the sentence and word level error rates as a function of penalty value. You should also try models where the first and second derivatives have been included: digits_mix_16_w_mic02.mmf or digits_mix16_m_mic02.mmf. Note that when using these, you will need to use the mic02.mfc configuration file. The same master label file "test_mw.mlf" should be used as the mic02.mfc features are derived from 12 MFCC parameters stored on the disk.

Testing with your voice

In this section, we will be testing the speech recognition system with your voice. This presents challenges due to the environment mismatch, microphone mismatch⁵ and possible difference between your accent and that of the people used to train the data base.

Generate 20 sentences from your grammar using HSLab. Then try using HTK in a live microphone mode:

```
HVite -H BestMasterMacroFile.mmf -C BestConfigFile.htk -w
      wdnet -p AppropriatePenalty dictionary
      digit_monophones
```

and see how well you do reading your 20 sentences. If there are other people in the lab, note whether or not you can observe differences in performance as the background noise varies.

What to turn in

For this assignment, proper execution of commands is 60 points. Due to the fact that the class elected to have an in-class Exam II as opposed to proctored one during the final exam slot, no formal report is expected. Instead, you should turn in the HTK summary tables and plots showing how the sentence and word accuracies vary with respect to the experimental parameters. These summaries are worth 30 points.

The final 10 points is related to your testing of the recognizer on your own voice. Provide the summary table and write a brief paragraph describing the types of errors that you may have seen. You may find it helpful to use the “-t” option with HResults which shows an aligned comparison of the recognized transcription and the labels. As an example, this would be the output of an insertion error (shown here on an utterance from TIMIT):

```
Aligned transcription:
/lab/speech/corpora/tidigits/mfc/test/woman/tb/3z9z322a_0 1.lab vs
/lab/speech/corpora/tidigits/mfc/test/woman/tb/3z9z322a_01.rec
LAB: THREE ZERO NINE          ZERO THREE TWO TWO
REC: THREE ZERO NINE EIGHT ZERO THREE TWO TWO
```

In addition to your abbreviated lab report, you should print out the commands that you used to conduct the experiment as well as their output. For commands that produce pages of output (e.g. running HResults with the -t option or hmm_train with verbose=true), please be kind to the trees and only include the first and last few lines of the output. Note that on UNIX/linux platforms, you can capture input/output with the script command (type man script for details).

⁵ In the literature, this is sometimes referred to as *transducer mismatch*.