CS4706 - Spring 2010
Project 2 - "Building a speech understanding system"

In this homework, you will design and build your own speech understanding system for the same TTS domain you chose in project 1. This system will be used to map an input spoken utterance to the concepts (degrees of freedom) in your domain. You should assume as input a grammatical English utterance (single sentence) in question form. The system will consist of two main components:

(a) An Automatic Speech Recognition (ASR) System: We provide a skeleton script that calls the ASR system using HTK (an HMM toolkit). The ASR acoustic models are trained on TIMIT, BDC, and the Columbia Games corpora.

- The **input** to this component is a wav file (audio format: mono, sampling rate: 16Khz)
- The **output** will be the automatic transcript in mlf file format (see an example below).
- To build this system you will provide a grammar that handles your input domain. This grammar is used by the ASR system to constrain the vocabulary when processing your input wav file.
- The base ASR system has been trained with different numbers of Gaussians per state in the HMM .You will need to experiment with these different HMMs to decide which works best in your domain.
- More information on the HTK toolkit, including the format of the grammar can be found here: (http://www.csie.ntu.edu.tw/%7Eb6506053/doc/htkbook.pdf)

(b) An Understanding Component:

- The **input** to this component is the ASR transcript produced in (a)
- The **output** will be a table that contains a (possibly partial) set of components that represents the query. You can structure this however you like with as many fields unspecified as makes sense in your domain. For example a reservation system may know about times, dates, destinations and origins, but you want to allow questions that reference only some of these items. You will need to include vocabulary that allows question answering.

For example, let's assume the your TTS domain involved train reservations and contained the following degrees of freedom:

- Departure city: New York, Boston, Baltimore, Newark, Jersey City, Washington, Albany, Poughkeepsie, Pittsburg, Philadelphia
- Destination city: (same set of cities as above)
- Departure day: Sunday, Monday, Saturday
- Departure Time: Morning, Noon, Afternoon, Evening, Night, Anytime

Here are three examples. Given the following utterances:

1. *What time can I travel from Boston to New York on Friday?*

The output concepts from your system should be:

Departure city	Boston
Destination	New York
Day	Friday
Time	UNSPECIFIED

2. *What trains leave from Washington on Monday evening?*

The output concepts from your system should be:

Departure city	Washington
Destination	UNSPECIFIED
Day	Monday
Time	Evening

3. *Is there a train from Washington to New York on Monday evening?*

The output concepts from your system should be:

Departure city	Washington
Destination	New York
Day	Monday
Time	Evening

You should create a grammar that covers as many different ways of asking questions **for your domain** as you can think of. The goal is to make your system as flexible as possible. However, your grammar should be limited enough so that the ASR perplexity is not so high as to affect performance. You should experiment to see what trade-offs between flexibility and performance work best. Part of the homework assignment is to see how well you can determine how much you can cover with reasonable performance. Note that your success will be judged on ***concept*** accuracy, not ***transcription*** accuracy.

Grammar format:

- Variables start with $ (e.g., $city)
- Terminals must be in capital letters (e.g., FRIDAY, TICKET)
- X Y is concatenation (e.g., I WANT)
- (X | Y) means X or Y – e.g., (WANT | NEED)
- [X] means optional, (e.g., [ON] FRIDAY)
- <X> Kleene closure (e.g., <$digit>) – comment: I do not think it would be used here

Here is an toy example of a grammar that covers the three examples above.

```
$city = BOSTON | NEWYORK | WASHINGTON | BALTIMORE;
$time = MORNING | EVENING;
$day = FRIDAY | MONDAY;

(SENT-START
       ((WHAT TRAINS LEAVE) | (WHAT TIME CAN I TRAVEL) | (IS THERE A TRAIN))
       (FROM|TO) $city (FROM | TO) $city ON $day [$time])
 SENT-END)
```

After designing your grammar you can run your ASR system as follows:

1. cd /proj/speech/users/cs4706/asrhw
2. mkdir USERNAME (e.g., fb2175)
3. cd USERNAME
4. mkdir /proj/speech/users/cs4706/asrhw/USERNAME/test/
5. Record five test utterances from your domain as wav files (16khz and mono) in Praat. For best recognition performance, leave ~1 second of silence at the beginning and ~1 second at the end of the file when recording. Call your files test[1-5].wav and save the files in /proj/speech/users/cs4706/asrhw/USERNAME/test/
6. Save the grammar to gram (not gram.txt) in /proj/speech/users/cs4706/asrhw/USERNAME
7. cd /proj/speech/users/cs4706/asrhw/USERNAME
8. Run: /proj/speech/users/cs4706/pasr/recognizePath.sh <your_wav_file> <hmm_file> <your_grammar_file>
   The script runs the recognizer on the given wav file and outputs a transcript (as out.mlf). The script takes three arguments.
   - A wav file containing the utterance
   - An HMM file specifying which model to use. You should experiment with different number of gaussians to determine which works best for your speaking voice and your domain. The possible models you can use are HMM_G1, HMM_G2, HMM_G4, HMM_G8, HMM_G12, HMM_G20 in the /proj/speech/users/cs4706/pasr/ directory.
   - A grammar file containing the grammar for your domain
9. Check the output of your recognizer in /proj/speech/users/cs4706/asrhw/USERNAME/out.mlf .

Now, you have a speech recognizer that takes a speech wav file and generates the transcript in *mlf* file format like the following:

```
#!MLF!#
"*/test2.rec"
5100000 5400000 I -250.811493
5400000 6300000 NEED -767.471863
6300000 7100000 TO -789.156311
7100000 9100000 GO -1631.608887
9100000 10000000 TO -913.183228
```

```
10000000 12400000 BALTIMORE -1923.127319
13300000 14000000 FROM -679.068176
14000000 14600000 WASHINGTON -560.649719
15900000 16500000 ON -547.398132
16500000 18500000 MONDAY -1689.119995
18500000 20200000 EVENING -1382.312256
```

Note: If your domain contains unusual words or proper nouns, you may have to modify the default dictionary. The dictionary is located at /proj/speech/users/cs4706/pasr/dict and is used in the recognizePath script. To add entries into the dictionary, you should copy it to your directory and make your own version of the recognizePath script to use the new dictionary.

Next, you must write a program that takes a wav file, runs the ASR System, and generates concept tables, as shown in examples 1 and 2 above. Each table should consist of concepts and their values separated by tabs, with one concept/value pair per line. Your scripts must be able to run on Speech Lab machines, so be sure that there are no version conflicts or other issues with your scripts. Test them before submission on a lab machine such as voix, veu, voce.

Example: Input test2.wav

@ RecognizeConcepts.sh ~/test/test2.wav

Output:

Departure city:	Boston
Destination:	New York
Day:	Friday
Time:	UNSPECIFIED

We will test your system on the wav files you provide for your domain as well other files we generate using questions covered by your grammar. You will be graded based on grammar coverage and concept accuracy.

Note: In Project 3 (upcoming) for the class, you will be creating a simple dialog system by combining the ASR and TTS systems for your domain. This may require modifications to your ASR grammar and to your TTS system. You should be thinking about now about how these systems will be combined in the future and what changes will be needed as you construct your grammar for the ASR.

Submission

You should submit the following:

1. A readme (txt or pdf) file file that
   - explains how to run your program, with a command line example.
   - explains the coverage of your grammar and any heuristics you employed, tradeoffs you made, or other interesting aspects of your approach.

- describes which set of HMMs works best for you and why you chose it.

2 The five input wav files in your voice that you've tested with your ASR system. N.B. if you are building this system with a partner, you should submit 10 files, 5 in each voice, asking different questions in all 10.

3. gram: a file containing your grammar in the format specified above
4. (optional) dict: a dictionary file if your domain requires it.
5. Your program that runs the ASR and extracts concepts (see @ above)

Upload these files in one zip file proj2-USERNAME.zip (e.g., fb2175.zip) to courseworks.