Massachusetts Institute of Technology
Department of Electrical Engineering & Computer Science

6.345 Automatic Speech Recognition
Spring, 2003

Issued: 03/21/03
Due: 04/09/03

Assignment 7
Fun With Graphical Models

Introduction

This assignment will give you some practice thinking about and using Bayesian networks (BNs), especially but not only for speech recognition. Part I consists of on-paper exercises; Part II is the "lab" part of this assignment, in which you will use the Graphical Models Toolkit (GMTK) to do speech recognition experiments. Parts I and II are independent, but you will probably find it helpful to work out the exercises in Part I first as a "warm-up" for Part II.

As in previous assignments, the following tasks (marked by **T**'s) should be completed during your lab session. The answers to the questions (marked by **Q**'s) should be handed in on the due date.

NOTE: For the lab part of this assignment, you will be better off working in the lab, where you can get face-to-face help from a TA if you need it. This is the first time this lab is being offered, so there may well be some kinks. Let us know if anything is unclear, and don't do the lab at the last minute! Note below that you are encouraged to confirm your answer to **Q12** with a TA before going on.

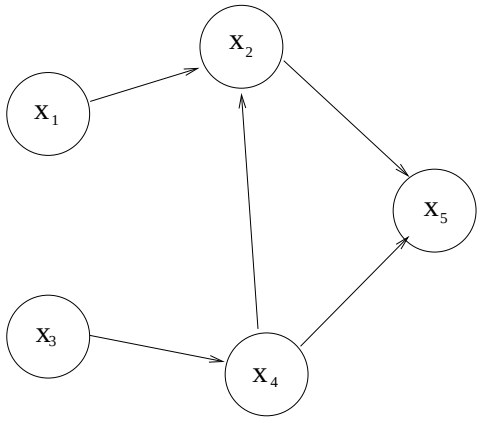
The end of this handout lists some references that you may find useful.

Part I: Warm-up Exercises

Notation: In this handout, we use the convention that capital letters represent random variables, while the corresponding lower-case letters represent those variables' values. We use the function $p(\cdot)$ as a shorthand for all types of probability functions (probability mass functions (PMFs) for discrete variables and densities (PDFs) for continuous variables), and use the convention that the lower-case arguments of the function identify the variables that it refers to. I.e., $p(x, y)$ is the PMF $P_{X,Y}(x, y)$ if X and Y are discrete and the PDF $f_{X,Y}(x, y)$ if they are continuous; and $p(x|y)$ means $P_{X|Y}(x|y)$ if X is discrete and $f_{X|Y}(x|y)$ if it is continuous.

A Simple Bayesian Network

Questions **Q1**–**Q3** refer to the following Bayesian network:



Q1: Write down the factored form of the joint probability $p(x_1, x_2, x_3, x_4, x_5)$ represented by this graph. (Note: This should take you about 5 seconds.)

Q2: Let the notation $A \perp B$ indicate that the variable A is independent of the variable B, and let $A \perp B|C$ indicate that A is independent of B given the variable C. For each of the following independencies, state whether or not it is implied by the above graph, and briefly explain why using the Bayes ball algorithm. If the independency is *not* implied by the graph, it suffices to give as explanation one path that the ball could follow.

- **(a)** $X_1 \perp X_4$
- **(b)** $X_1 \perp X_4|X_5$
- **(c)** $X_3 \perp X_5$
- **(d)** $X_3 \perp X_5|X_2$

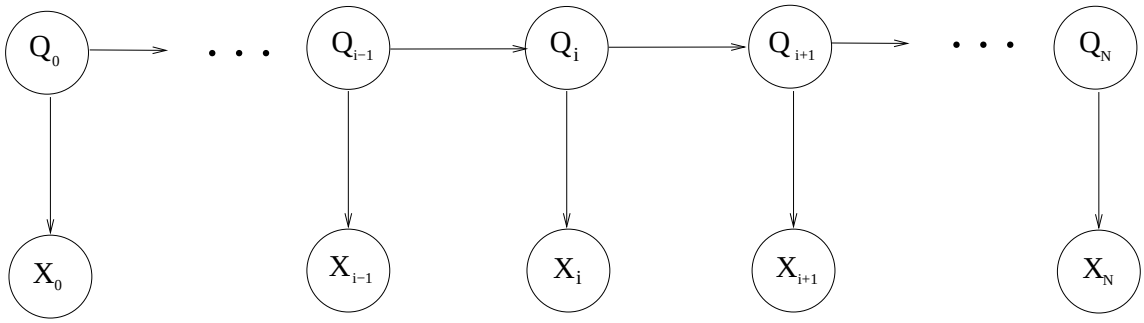
Q3: Suppose that $X_1, X_2, \ldots, X_5$ are all binary variables.

- **(a)** How many parameters are required to completely specify the joint distribution $p(x_1, \ldots, x_5)$?
- **(b)** How many parameters would be required if there were an additional edge from X_1 to X_5?
- **(c)** How many parameters would be required if we had *no* information about the independence relationships among the variables? Also, draw *two* Bayesian networks for the variables $X_1, \ldots, X_5$ that would represent such a situation, and give the factorization of $p(x_1, \ldots, x_5)$ that is represented by each of your BNs.

Q4: (This question does not relate to the figure.) By definition, the graph of a BN must have no directed cycles. We can start to see why this is needed by considering what would happen if we removed this restriction and looking at some graphs that do contain cycles. Give one example of a graph that contains a directed cycle, and show *mathematically* that trying to interpret it using Bayesian network semantics of probability factorization results in a contradiction.

A Dynamic Bayesian Network

Questions **Q5**–**Q8** refer to the dynamic Bayesian network (DBN) below and Tables 1–3.



Assume that, in each frame, Q_i is discrete with cardinality 4, X_i is continuous and one-dimensional, and $p(x_i|q_i)$ is a mixture of up to M_{q_i} Gaussians, i.e.

$$p(x_i|q_i) = \sum_{m=1}^{M_{q_i}} w_{m,q_i} \mathcal{N}(x_i; \mu_{m,q_i}, \sigma^2_{m,q_i}) \tag{1}$$

Q5: Suppose this DBN is "unrolled" out to 3 frames (i.e. $N = 2$). Write down the factored expression for $p(x_0, x_1, x_2, q_0, q_1, q_2)$ represented by this graph.

Q6: This DBN represents a hidden Markov model. For each of the conditional probability tables in Tables 1–3, draw the state transition diagram for the HMM to which it corresponds.

Q7: For one of the tables above, there is a simpler DBN that could represent the same distribution. Which table is it, and what would the simplified DBN look like in that case?

Q8: Suppose we wanted to implement our DBN using a software package in which continuous variables can only have Gaussian distributions (i.e., mixtures of Gaussians are not possible). Describe how we can still represent the same joint distribution over $X_1, \ldots, X_N, Q_1, \ldots, Q_N$ as our original DBN by adding to the graph one more variable (and all the necessary dependencies). Draw the resulting modified DBN, and specify the cardinality of the new variable, its local probability function, and the local probability functions of any children it has, in terms of quantities already defined.

q_{i-1} \ q_i	0	1	2	3
0	0.2	0.4	0.1	0.3
1	0.2	0.4	0.1	0.3
2	0.2	0.4	0.1	0.3
3	0.2	0.4	0.1	0.3

Table 1: $p(q_i|q_{i-1})$, version 1.

q_{i-1} \ q_i	0	1	2	3
0	0.9	0.1	0	0
1	0	0.9	0.1	0
2	0	0	0.9	0.1
3	0	0	0	1

Table 2: $p(q_i|q_{i-1})$, version 2.

q_{i-1} \ q_i	0	1	2	3
0	0.5	0	0	0.5
1	0.2	0	0.4	0.4
2	0	0.3	0	0.7
3	0	0	1	0

Table 3: $p(q_i|q_{i-1})$, version 3.

Part II: Using GMTK for Speech Recognition on the Aurora Task

In this part you will carry out speech recognition experiments using the Graphical Models Toolkit (GMTK) [1] on a small subset of the Aurora 2.0 corpus, which contains connected digit strings with various amounts and types of added noise (subway noise, babble, etc.). You will learn some of the features of the toolkit and use it to represent, train, and test the recognizers.

Note: In this handout, commands that you should type are printed as separate lines in `typewriter` font, with a "%" to indicate the UNIX prompt. When a command extends beyond the width of the page, it is printed over multiple lines, though you should type it as one.

Getting Started

To get started for this lab, type the following at the UNIX prompt:

```
% start_lab7.cmd
% cd lab7
```

`start_lab7.cmd` will create a new subdirectory under your home directory called `lab7` and populate it with the files needed for this lab. These files contain data, models, and parameter specifications used during the lab. Descriptions of the files are given mainly as comments within the files themselves. In the first part of the lab, we will look at some of these files in detail to

[1] J. Bilmes and G. Zweig, "The Graphical Models Toolkit: An Open Source Software System for Speech and Time-Series Processing." *IEEE International Conference on Acoustics, Speech, and Signal Processing*, June 2002, Orlando, Florida.

learn a little about how GMTK represents DBN structures and parameters. All of the path names mentioned are with respect to the `lab7` directory (e.g. the filename `foo` refers to `~/lab7/foo`).

GMTK Basics

We will start with some brief background to get you acquainted with the features of GMTK that you will need (and perhaps a couple that you will not need, but that are good to know about). As we look at the features of the toolkit, we will train and test a baseline recognizer. You will then make some modifications to the recognizer and run experiments with different configurations.

In GMTK, the structure of a model (i.e. the variables and dependencies) is defined in a *structure file*. The structure file also contains certain attributes of each variable (hidden vs. observed, discrete vs. continuous, etc.) and a name and type for each variable's local probability function (Gaussian, probability table, deterministic, etc.). Technically, then, it actually gives more than just the structure of the model. The parameters for each local probability are then specified in separate *parameter files*. We'll look at each type of file separately.

The two main GMTK programs that you need to be aware of are `gmtkViterbi` and `gmtkEMtrain`. `gmtkViterbi` takes in a structure, parameters, and *observation files* giving the values of all observed variables, computes the maximum-likelihood settings of the hidden variables, and outputs the values of the variables of interest (which, for speech recognition, is usually the word). `gmtkEMtrain` takes in a structure, initial parameter settings, and observation files containing the training observations, and runs the EM algorithm to find the maximum-likelihood settings of the parameters. For most of this lab, you will not be directly running `gmtkEMtrain`, but rather a wrapper script that uses `gmtkEMtrain` to do training in parallel over multiple machines.

Structure Files

The DBN structures that we will use for training and decoding a baseline Aurora recognizer are defined in `STRUCTURES/[training,decoding].str`. This baseline recognizer is an HMM-based recognizer, with the phone variable being the HMM state.

Read through the structure files to familiarize yourself with the GMTK structure specification syntax and the particular structure we are working with. The decoding structure is the same as the one shown in class (see the lecture slides on the course web page). The training structure is a bit different, as it must take into account the fact that during training we know the word string (but not which frames go with which words). Look at the decoding structure file first, as it introduces most of the syntax.

Q9: Draw the structure of the training DBN, including all of the frames/variables that are in the structure file. In your diagram, use shaded nodes for observed variables and dotted arrows to indicate switching parents. You may find it easiest to start with the decoding structure from the lecture slides and modify it as needed. Note that the observation variable is multi-dimensional, but is still treated as a single variable in this structure; it just happens to be a vector variable (in this case, of MFCC values). Denote it as a single node in your graph.

Also indicate, for each variable in each frame (either next to its node on the graph or in a separate list), whether it is discrete or continuous, and, if it is discrete, whether it is deter-

ministic or not. (Yes, this is just an exercise in reading the file! But it will be easier later to refer back to your drawing than to the file.)

Q10: If this is an HMM, then why does this model not just have two variables, a state and an observation? Could we have implemented the same functionality using a model with just these two variables? If so, how?

Parameter Files

In GMTK, continuous variables can have Gaussian or Gaussian mixture distributions. The probability of a discrete variable is given via conditional probability tables, or CPTs. In general, a CPT is just a (multidimensional) table of probability values, one row per combination of parent values. The parameters of Gaussians and the probability tables can be trained via EM and are stored in a *trainable parameters file*.

For our baseline recognizer, the initial trainable parameter file, containing the parameters that you will use to initialize EM training, are found in `PARAMS/flat_start.gmp`. Examine this file, which is also documented via comments. In this case, we are initializing all of the Gaussians to $\mathcal{N}(0, 10)$ and all of the probability tables to uniform probabilities.

For the special case of *deterministic* variables, i.e. those that are just functions of their parents, one can specify a "deterministic CPT", which simply maps from a combination of parent values to a single value using a decision tree. (Of course, one can always use a non-deterministic CPT for a deterministic variable, but the table would contain almost all zeros!). In such a decision tree, each of the branching points queries the value of one of the variable's parents, and based on that descends down the appropriate branch. Each path in the decision tree corresponds to some combination of values of the parent variables. The leaf nodes of the tree, i.e. the "answers", give the values of the child variable for the corresponding paths down the tree. The "parameters" for deterministic variables, i.e. the specifications of their deterministic CPTs and the the decision trees they use, are given in a separate file.

The fixed parameters for the baseline recognizer are found in `PARAMS/masterFile.template`. Again, the syntax of the file is given via comments in the file itself. (It is called a "master file" because it actually has some further capabilities, which we will not be using; and it is called a "template" only because it will be used to generate multiple files during the parallel training.)

File lists, feature files, waveforms

In these experiments you will deal with a training set of 100 utterances, consisting of two subsets of 50 utterances each. One of the subsets is a clean recording, and the other has had subway noise added to it with a signal-to-noise ratio (SNR) of 5 dB. The test sets also consist of 50 utterances at each noise level.

The files `train.filelist`, `test.clean.filelist`, and `test.SNR5.filelist` contain lists of the files in which the acoustic observations (in this case, MFCCs plus their derivatives and second derivatives) are stored. The file `train.wavlist` contains a list of the waveform files in the training set. If you are curious, you can listen to a few of these to get a sense of the type

of speech we are dealing with and the difference between the two noise levels. You can play a waveform file by running

```
% waveform_play -normalize <filename>
```

Training the baseline recognizer

T1: Carry out 4 iterations of the EM algorithm on the baseline structure by running

```
% emtrain_parallel header.emtrain.baseline
```

`emtrain_parallel` runs each iteration on multiple machines in parallel using the SLS group's *host dispatch* facility. The EM algorithm is parallelized by splitting up the "E" and "M" steps; the "E" step is done by splitting up the training set into a number of smaller chunks and computing the statistics of the chunks on different machines in parallel, while the "M" step collects all of the statistics (on a single machine) and performs the maximization.

The argument to `emtrain_parallel` is a header file, in which we specify various parameters needed for training such as the locations of the various files, the number of iterations, and so on. As indicated in the header file, the final parameters will be written to `PARAMS/learned_params.baseline.gmp`.

When running a parallel job on the host dispatch, you can get some information on the status of all hosts, as well as of current jobs, by running `phd -status`. The training run should take one to a few minutes, depending on how many other jobs are running at the same time.

T2: Use the baseline recognizer to decode the utterances in the two test sets by running

```
% gmtkViterbi -of1 test.clean.filelist -strF STRUCTURES/decoding.str
-inputMaster PARAMS/masterFile.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/learned_params.baseline.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition | tee
OUT/clean.baseline.out

% gmtkViterbi -of1 test.SNR5.filelist -strF STRUCTURES/decoding.str
-inputMaster PARAMS/masterFile.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/learned_params.baseline.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition | tee
OUT/SNR5.baseline.out
```

These commands are also stored in the file `decode_commands.baseline`, so that you do not have to type all of the above by hand. To find out more information about the command-line arguments, type `gmtkViterbi` without any arguments.

The decoding is done locally (i.e. on your machine and not in parallel), and each decoding run should take about a minute. The output will go to both the screen and the files `OUT/[clean,SNR5].baseline.out`. Here is an example of what the output should look like for two utterances:

```
Example prob: -31068.1 : 166 frames
sil (0-32)
five (33-62)
nine (63-99)
seven (100-136)
sp (137-165)
Example prob: -35347 : 187 frames
sil (0-20)
one (21-42)
six (43-84)
sil (85-103)
three (104-135)
oh (136-158)
sil (159-186)
```

The first line gives the total log probability of the acoustic observations given the model, as well as the number of frames in the utterance. The remaining lines show the decoded word string, with the frame numbers corresponding to each word in parentheses. In the first utterance above, the first decoded word is silence, extending from frame 0 to frame 32; the second is "five", from frame 33 to frame 62; and so on.

Measure the error rates on these sets by running

```
% score_aurora_SubsInsDel OUT/clean.baseline.out test.clean.filelist
word_list

% score_aurora_SubsInsDel OUT/SNR5.baseline.out test.SNR5.filelist
word_list
```

This will output information on the substitution, insertion, and deletion rates, the total word error rate, and detailed counts of each type of error.

T3: Also train separate recognizers for the two noise conditions, i.e. train one recognizer on only the 50 clean utterances and one on the 50 noisy utterances:

```
% emtrain_parallel header.emtrain.clean

% emtrain_parallel header.emtrain.SNR5
```

The trained parameters will be saved in `PARAMS/learned_params.clean.gmp` and `PARAMS/learned_params.SNR5.gmp`, respectively.

T4: As above, test each of these two recognizers on both the clean and the SNR5 test sets. The commands you should run for this are in the file `decode_commands.separate`.

Q11: Tabulate the word error rates you measured in **T2** and **T4**. Does the recognizer perform better when decoding on utterances with the same SNR as it was trained on than when it is trained on the mixed training set? Was this result inevitable? Why or why not? (Note that this question has nothing specifically to do with graphical models.)

Adding a noise variable

We would now like to modify the baseline structure so as to explicitly model the fact that each utterance is either clean or noisy, using different Gaussian mixture models for each condition. We would like to do this by adding a binary variable, called `noise`, which will take on the value 0 for clean utterances and 1 for noisy utterances. The observation will now depend on *both* the phone state and the noise value, i.e. it will have a different Gaussian mixture for each combination of phone state and noise value. During training, the noise variable will be observed (since we know which training utterances belong to which noise condition); during testing it will be hidden. This can be viewed as a form of implicit noise classification, since the value of the noise variable during decoding will tell us which noise condition the recognizer thinks the utterance corresponds to.

We will assume that this new recognizer will be tested on test data with the same distribution of clean vs. noisy utterances as the training set.

A new trainable parameters file, `PARAMS/flat_start.noise.gmp`, has been provided for use with this new model; since there are now two separate Gaussian mixtures per phone state, this parameters file has twice as many Gaussian mixtures, Gaussian components, means, etc. as `PARAMS/flat_start.gmp`.

The file `train.noise.filelist` contains a list of files with noise level "observations" for each of the training utterances. For a clean utterance, its observation file simply contains a "0" for each frame; for a noisy utterance, it contains a "1" for each frame. (Theoretically we shouldn't need a separate value for each frame, since we know that the noise condition is the same throughout an utterance, but GMTK requires that all frames have the same number of observations.)

Q12: Draw a new pair of training and decoding structures that represent this new model, by augmenting the baseline training/decoding graphs with the noise variable and any necessary additional dependencies. Also specify what the local probability functions are for the noise variable and for any other variables whose local probabilities are different in the new structure. (We've already said, for example, that the observation variable will depend on both the phone state and the noise.) For any deterministic variables, you can specify the local probability as a rule or a function (e.g. "v = the value of the i^{th} parent", or "v = 0 if the parent is 1, and 1 otherwise").

You are strongly encouraged to confirm your answer to this question with a TA before going on, as you will next be implementing this model.

T5: We would now like to modify the necessary files to implement this new model. You have been provided partial files, which you will need to complete.

Do not be discouraged if this step takes a long time... Be careful and patient, and if you get stuck, feel free to ask for guidance.

First, take a look at `STRUCTURES/training.noise.str`. The observation variable has already been modified appropriately for the new model, and the skeleton of a new variable called "noise" has been included. You will need to fill in the remainder of the specification for the noise variable in each frame. You can name any new local probability functions any way you like, as long as you are consistent.

Next, you will need to modify the initial trainable parameters file `PARAMS/flat_start.noise.gmp` and/or the non-trainable parameters in

PARAMS/masterFile.noise.template. Both of these files already include the modifications needed for the observation variable (new Gaussians and a decision tree to map from the phone state and noise to the corresponding Gaussian mixture). You will need to add any other new CPTs to these files in the appropriate places. Remember to use the same names for the new CPTs as you specified in the structure file. Also, don't forget to increment the number of CPTs when you add a CPT!

Finally, you will need to do the same for the decoding files STRUCTURES/decoding.noise.str and PARAMS/masterFile.noise.decode.

When you think you are done making the necessary changes, you should make sure that your model runs. To test the training model, run

```
% convert_masterfile.pl < PARAMS/masterFile.noise.template >
PARAMS/masterFile.noise

% gmtkEMtrain -of1 train.local.filelist -nf1 42 -ni1 0 -fmt1 ascii
-of2 train.noise.local.filelist -nf2 0 -ni2 1 -fmt2 ascii
-inputMaster PARAMS/masterFile.noise -inputTrainable
PARAMS/flat_start.noise.gmp -binInputTrainable false -strFile
STRUCTURES/training.noise.str -maxE 1 -trrng 0:1 -baseCaseT 100000
```

You can also find these commands in train_command, so that you don't have to type them in. This will run gmtkEMtrain on two utterances for one iteration. The last line of output should include "PROGRAM ENDED SUCCESSFULLY WITH STATUS 0". If there is an error, go back and fix your structure and parameter files and repeat.

To test the decoding model, run

```
% gmtkViterbi -of1 test.clean.filelist -strF
STRUCTURES/decoding.noise.str -inputMaster
PARAMS/masterFile.noise.decode -nf1 42 -fmt1 ascii
-inputTrainable PARAMS/flat_start.noise.gmp -beam 500
-printWordVar word -varMap word_map -trans wordTransition
-baseCaseT 100000 -dcdrng 0:1
```

(Or, equivalently, decode_command). This will run gmtkViterbi on two utterances, using the initial parameters. Successful completion is again indicated by the above phrase.

T6: Once your model runs, train it on all 100 training utterances as before. A new header file that uses the structure and parameter files has been provided. To train the recognizer, run:

```
% emtrain_parallel header.emtrain.noise
```

Also test your recognizer's performance on the two test sets, using the commands in decode_commands.noise. Score the outputs using

```
% score_aurora_SubsInsDel OUT/clean.noise.out test.clean.filelist
word_list

% score_aurora_SubsInsDel OUT/SNR5.noise.out test.SNR5.filelist
word_list
```

Q13: What are the word error rates on both sets? How do these compare with your previous results? Try to explain any differences between the performance of this model and that of the models trained on each noise condition separately. (Hint: Think about how this model differs from having a *complete* separate model for each condition.)

T7: To test how well the new recognizer does at classifying the test utterances as clean or noisy, run the commands in `decode_commands.noise_classify`. The output for each utterance will now be the value of the noise variable during each word (which should be the same for all words in an utterance). The output will be stored in `OUT/[clean,SNR5].noise_class.out`.

Q14: How many noise classfication errors does your model make in each condition?

Q15: Could we have implemented the same model using an HMM-based recognizer? If so, how? If not, why not? (By the "same model", we mean one that would always behave exactly the same; in particular, it should give the same decoding outputs.)

Adding a hidden auxiliary variable

Now we would like to find out what would happen if we added a binary variable to the baseline recognizer as before, but didn't "tell" the recognizer that this additional variable was supposed to represent the noise level. That is, we would like to keep it hidden during both training and testing, and allow the EM algorithm to train the parameters as it pleases.

T8: Repeat **T5**, this time for a model implementing this new auxiliary variable. Fill in the training and decoding structure files `STRUCTURES/[training,decoding].aux_var.str`, trainable parameters file `PARAMS/flat_start.aux_var.gmp`, and non-trainable parameters file `PARAMS/masterFile.aux_var.[template,decode]`. Note that the decoding structure and parameters can be exactly the same as in **T5**. In fact, if you wish you can just copy your `STRUCTURES/decoding.noise.str` and `PARAMS/masterFile.noise.decode`, changing the variable name from "noise" to "aux" and making sure that the CPT names are consistent between training and decoding.

You need not submit a diagram of your new training structure, but you may find it useful to draw one before editing the structure file.

You can again test that your new models run before training, using `train_command.aux_var` and `decode_command.aux_var`.

T9: Train your model using

```
% emtrain_parallel header.emtrain.aux_var
```

Decode with the trained model using `decode_commands.aux_var`, and measure the error rates using

```
% score_aurora_SubsInsDel OUT/clean.aux_var.out test.clean.filelist
word_list

% score_aurora_SubsInsDel OUT/SNR5.aux_var.out test.SNR5.filelist
word_list
```

Also, look at the values of the auxiliarly variable for each utterance using `decode_commands.aux_classify`. The outputs will be in `OUT/[clean,SNR5].aux_class.out`. The format will be the same, except that this time the output values will be 0 or 1, rather than "clean" or "noisy".

Q16: Does this new auxiliary variable seem to have learned to associate itself with the noise level? What does an auxiliary value of "0" correspond to, and what does a "1" correspond to? What do you think would happen if we ran the same experiment, except used only clean speech as training data (i.e. how might the values of the auxiliary variable pattern in that case)?

Q18: (Extra credit!)

(a) If you look at the Gaussian means in `flat_start.aux_var.gmp`, you will notice that they are not uniform (as were the ones we used for the baseline model and the model with the observed noise variable). This parameters file was initialized with random values rather than uniform ones. Can you explain why this is necessary in this case? You may want to try training this recognizer with a uniform initialization instead (by modifying `flat_start.aux_var.gmp` to use the Gaussian means from `flat_start.noise.gmp`), and see what happens.

(b) Based on your answer to part (a), can you explain any differences between the word error rates you found in **T9**, using the hidden auxiliary variable, and in **T6**, using the noise variable?

References

You may find the following readings useful. There are links to them on the course web site.

[1] J. Bilmes, "Graphical Models and Automatic Speech Recognition." In *Mathematical Foundations of Speech and Language Processing, Institute of Mathematical Analysis Volumes in Mathematics Series*, Springer-Verlag, 2003.

This paper gives an overview of graphical models in general, as well as of their uses in speech recognition. The most relevant sections, for our purposes, are Sections 1, 2.1-2.5 (although the details of inference in 2.5 can be skipped), 3.4, 3.9, and 4-6.

[2] G. Zweig, *Speech Recognition with Dynamic Bayesian Networks.* Ph.D. dissertation, U.C. Berkeley, 1998.

This thesis develops the use of DBNs for speech recognition. Chapter 6 is the main chapter that discusses how DBNs can be configured for ASR.

[3] J. Bilmes, "What HMMs can do." UWEETR-2002-0003 (U. Washington Technical Report), Feb, 2002.

This paper goes more deeply into why models beyond HMMs may be needed for speech recognition. It shows that HMMs are extremely powerful in theory, but that other models may be better under real-world constraints of finite data and computation. This is not required reading by any means, but you may find it interesting.