Massachusetts Institute of Technology
Department of Electrical Engineering & Computer Science

6.345 Automatic Speech Recognition
Spring, 2003

Issued: 02/28/03
Due: 03/12/03

Assignment 4
Acoustic Phonetics, Acoustic Modeling, and Pattern Classification

---

Introduction

The purpose of this assignment is to strengthen your understanding of acoustic phonetics, acoustic modeling, and pattern classification. You will first investigate the acoustic properties of the fricative sounds of English [s z š ž f v θ ð] by examining acoustic measurements taken from examples of these phones extracted from the TIMIT corpus. Next, you will investigate several different techniques for modeling these measurements, and then perform a set of pattern classification experiments using these techniques.

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.

Getting Started

Assignment 4 will be performed using MATLAB®, a mathematical software package created by MathWorks, Inc. This handout contains sufficient documentation to complete the laboratory exercises. Additional information is available online via the `help` command within MATLAB®. To start the lab, type the following at the UNIX prompt:

```
% start_lab4.cmd
```

This command will start up MATLAB®. After initialization, the MATLAB® command-line prompt will appear. At the prompt, type the following command:

```
>> init_lab4
```

This command will load in the data that you will use during the assignment.

Description of the Data

The data used in this assignment was extracted from the TIMIT corpus. A set of eight different phones, the fricatives [s z š ž f v θ ð], were chosen for analysis in this assignment. In MATLAB® the labels 's', 'z', 'sh', 'zh', 'f', 'v', 'th', and 'dh', respectively, are used to represent these eight phones. An observation vector of six measurements was created for every example of these phones in the corpus. The six measurements are:

- **MFCC0**: The average value of the zeroth MFCC coefficient.
- **MFCC1**: The average value of the first MFCC coefficient.
- **TOT Energy**: The average total log energy.
- **LF Energy**: The average log energy between 50 Hz and 200 Hz (low-frequency energy).
- **MFR Energy**: The ratio of the log energy between 2 kHz and 4 kHz and that between 2 kHz and 6 kHz (mid-frequency ratio energy).
- **Duration**: The log duration of the phone.

Different groupings of the above phones have been created to facilitate analysis during the assignment. After the Assignment 4 data has been loaded, type the following command:

```
>> whos
```

This command will list the MATLAB variable names for each piece of data used in the assignment. The following is a description of the variables:

- For Part I of the assignment:
  - **fric_AP**: A set of vectors for the phones [s z š ž] (where AP refers to alveolar and palatal). Each row in the matrix corresponds to an instance of a phone, and the columns contain the feature measurements for each instance.
  - **fric_AP_index**: A set of indices indicating the "phone class" that each row vector in **fric_AP** is associated with. These labels partition the data into two classes: alveolar and palatal. This matrix also contains information about the left and right context phones.
  - **fric_SW**: A set of vectors for the phones [s š f θ] (SW = strong/weak).
  - **fric_SW_index**: A set of indices indicating the "phone class" that each row vector in **fric_SW** is associated with. Partitions the data into two classes: strong and weak.
  - **fric_VU**: A set of vectors for the phones [z v s f] (VU = voiced/unvoiced).
  - **fric_VU_index**: A set of indices indicating the "phone class" that each row vector in **fric_VU** is associated with. Partitions the data into two classes: voiced and unvoiced.
- For Parts II and III of the assignment:
  - **train**: The set of training vectors for the phones [z š f ð]. Each row in the matrix is a phone instance and the columns contain the feature measurements.

  - **train_index**: A set of indices indicating the "phone class" that each row vector in **train** is associated with. Also contains information about the left and right context phones.
  - **test**: The set of test vectors for the phones [z š f ð].
  - **test_index**: A set of indices indicating the "phone class" that each row vector in **test** is associated with.
  - **train_**<*label*>: Each set of this form contains only the training vectors for a particular phone, e.g., **train_z** contains the training vectors for the phone [z].
  - **train_**<*label*>**_index**: A set of indices indicating the "phone class" that each row vector in **train_**<*label*> is associated with.
  - **train_**<*N*>: Each set of this form contains a subset of the training vectors. $N = 1$ has 77 vectors, $N = 2$ has 152 vectors, $N = 3$ has 302 vectors, and $N = 4$ has 601 vectors.
  - **train_**<*N*>**_index**: A set of indices indicating the "phone class" that each row vector in **train_**<*N*> is associated with.

- Additional variables:
  - **measurements**: A vector of strings containing a label for each feature measurement. This is used internally by some plotting routines to label the axes.
  - **phone_labels**: A vector of strings containing a label for each phone index. This is passed into some plotting routines if you want to label individual data points.
  - **phone_colors**: A vector of strings indicating the color and symbol type to use for plotting each phone listed in **phone_labels**. This is used internally by some plotting routines to color points and text.

To observe the contents of any of these variables simply enter the name of the variable at the MATLAB prompt. For example, for the variable **measurements**:

```
>> measurements

measurements =

MFCC0
MFCC1
TOT Energy
LF Energy
MFR Energy
Duration
```

Because many of the variables contain extremely large amounts of data, you may wish to observe only small portions of the entire data at a time. You can do this by expressing a range to be viewed. For example, for the variable **train** we can view the first three vectors of the set with the command:

```
>> train(1:3,:)

ans =

    8.8863   -1.6315    7.7928    5.5401   17.6856    8.0378
    7.8264   -1.2994    7.3976    4.8848   16.4082    8.1038
    9.0345   -2.1741    8.4573    5.8948   16.9312    7.3185
```

The following is a table of the IPA symbols for the phones and the corresponding TIMIT labels along with example occurrences. This is provided as a reference so that you can understand what the TIMIT phone labels mean when you see them in the assignment plots.

IPA	TIMIT	Example	IPA	TIMIT	Example
[ɑ]	aa	b*o*b	[ɨ]	ix	deb*i*t
[æ]	ae	b*a*t	[i]	iy	b*ee*t
[ʌ]	ah	b*u*t	[ǰ]	jh	*j*oke
[ɔ]	ao	b*ou*ght	[k̟]	k	*k*ey
[ɑʷ]	aw	b*ou*t	[k̟□]	kcl	k closure
[ə]	ax	*a*bout	[l̟]	l	*l*ay
[əʰ]	ax-h	p*o*tato	[m]	m	*m*o*m*
[ɚ]	axr	butt*er*	[n]	n	*n*oo*n*
[ɑʸ]	ay	b*i*te	[ŋ]	ng	si*ng*
[b̟]	b	*b*ee	[ɾ̃]	nx	wi*nn*er
[b̟□]	bcl	b closure	[o]	ow	bo*a*t
[č]	ch	*ch*oke	[ɔʸ]	oy	bo*y*
[d̟]	d	*d*ay	[p]	p	*p*ea
[d̟□]	dcl	d closure	[□]	pau	pause
[ð]	dh	*th*en	[p□]	pcl	p closure
[ɾ]	dx	mu*dd*y	[ʔ]	q	glottal stop
[ɛ]	eh	b*e*t	[r]	r	*r*ay
[l̩]	el	bott*le*	[s]	s	*s*ea
[m̩]	em	bott*om*	[š]	sh	*sh*e
[n̩]	en	butt*on*	[t]	t	*t*ea
[ŋ̩]	eng	Wash*ing*ton	[t□]	tcl	t closure
[̩]	epi	epenthetic silence	[θ]	th	*th*in
[ɝ]	er	b*ir*d	[ʊ]	uh	bo*o*k
[e]	ey	b*ai*t	[u]	uw	bo*o*t
[f̟]	f	*f*in	[ü]	ux	t*oo*t
[g̟]	g	*g*ay	[v]	v	*v*an
[g̟□]	gcl	g closure	[w]	w	*w*ay
[h̟]	hh	*h*ay	[y]	y	*y*acht
[ɦ̟]	hv	a*h*ead	[z]	z	*z*one
[ɪ]	ih	b*i*t	[ž]	zh	a*z*ure
-	h#	utterance initial and final silence			

Functions and Commands

To examine and manipulate the data, a locker of useful acoustic modeling and pattern classification commands has been created. To obtain a list of these commands, type:

```
>> help assign4
```

To read a manual page description of any of these commands, type:

```
>> help <command_name>
```

Saving Plots as Postscript Files

You can save the current plot displayed in the MATLAB® figure window by using the `print` command:

```
>> print -deps matlab_plot.eps
```

This will save the current plot in Encapsulated PostScript format to the file `matlab_plot.eps` For more information about the `print` command, type `help print` at the MATLAB® prompt.

Multiple Plot Windows

By default, MATLAB® uses one "figure window" for drawing plots. You can create additional figure windows to display multiple plots simultaneously by using the `figure` command. Type `help figure` at the MATLAB® prompt for more information about how to use this command.

Alternatively, if you want to generate multiple "subplots" within one figure window, you can use the `subplot` command. Type `help subplot` at the MATLAB® prompt for more information about how to use this command.

Part I: Acoustic Phonetics

Understanding the properties of acoustic measurements and their relationship to the different phones is an important first step in determining how to model the features/measurements and what approach to pattern classification should be used.

In this part of the assignment, you will investigate the acoustic properties of the fricative sounds of English [s z š ž f v θ ð]. We will examine

- how phonetic contrasts may be related to various acoustic measurements, and
- the influences of context on the acoustic properties of fricative speech sounds.

Before beginning this part of the assignment, we recommend that you reacquaint yourself with the general acoustic-phonetic properties of fricatives. Please refer to the lecture notes on acoustic phonetics.

Strong vs. Weak Fricatives

We will begin our investigation by examining the acoustic correlates of the feature *strident*. The strident or strong fricatives are [s z š ž], and the non-strident or weak fricatives are [f v θ ð]. For this task, we will use the **fric_SW** data set, which consists of the fricatives [s š f θ] partitioned into two classes: strong and weak.

T1: Generate histogram plots (using `plot_histograms`) of this data set for each of the six different measurements and examine their abilities to separate the two classes. For example, to create a histogram plot using the first feature measurement (**MFCC0**), use the following command:

```
>> plot_histograms(fric_SW,1,fric_SW_index);
```

For the **average total energy** measurement, the overlaid histograms for the two classes should reveal significantly different distributions for strong *vs.* weak fricatives. However, there should be some overlap between them. We will now take a closer look at some of the unusually strong *weak* fricatives whose **average total energy** puts them in the overlap region. This is best accomplished using scatter plots.

T2: Generate a two-dimensional scatter plot using the `plot_clusters` command for the **fric_SW** data. For example, to generate a scatter plot of the first two dimensions of the data, use the following command:

```
>> plot_clusters(fric_SW,1,2,fric_SW_index);
```

Scatter plots enable you to see how multiple measurements interact with each other. In this plot, each phone occurrence is represented by a colored symbol at the appropriate coordinates. Explore different pairs of measurements and examine their abilities to discriminate between the two classes. Notice how the combined use of two measurements can lead to better separation of the two classes than using a single measurement alone.

T3: Redraw the scatter plot using the two most discriminating measurements derived from **T2** as the plot dimensions. One of them should be **average total energy**. This time, also plot the phone labels of the data points by including `phone_labels` in the call to `plot_clusters` as follows (this plot may be a bit slow to generate):

```
>> plot_clusters(fric_SW,2,3,fric_SW_index,phone_labels);
```

The plotted labels include information not only about the identity of the specified "middle" phone, but also about the identity of its left and right neighbors. The labels have the following format: `left_middle_right`.

Zoom in (by clicking the left mouse button over the region of interest in the figure; click the middle button to zoom out) on the overlap region where the weak fricatives have large **average total energy**. You may also want to resize the plot to get a better view. Examine the phonetic contexts of these weak fricatives.

Q1: Can you identify a phonetic environment in which unusually strong [f θ]'s appear? For this and all subsequent similar questions, if you are having trouble determining the relevant contexts from the scatter plot alone, try to come up with some ideas based on what you know from lecture about acoustic phonetics, and test your hypotheses by looking at the scatter plot. If there are any contextual effects that you think should be there but are not evident in the scatter plot, try to describe the effects and why they may not be evident.

Voiced vs. Unvoiced Fricatives

We will follow a similar procedure to contrast the distributions of the *voiced* [z ž v ð] and *unvoiced* [s š f θ] fricatives. For this task, we will use the **fric_VU** data set, which consists of the fricatives [z v s f] partitioned into two classes: voiced and unvoiced.

T4: Using the **fric_VU** data set, examine histogram plots (using `plot_histograms`) for each of the six different measurements and examine their ability to separate the two classes.

Notice that while the distributions of **duration** for the voiced and unvoiced fricatives are distinguishable, there is also substantial overlap between them. This is due to the fact that acoustic properties of phonemes are affected by the environments in which they appear.

T5: Generate a two-dimensional scatter plot using the `plot_clusters` command for the **fric_VU** data using the **duration** and **low-frequency energy** measurements. Include the phone labels in the plot.

Q2: From the set of available contexts provided in the data sample, try to determine two factors that have a significant influence on fricative durations, i.e., what contexts can cause a fricative to be lengthened? To be shortened?

T6: Voiced and voiceless fricatives also differ in the amount of low-frequency energy. Examine this acoustic difference using the same procedure as outlined above, using the **low-frequency energy** measurement of energy from 50 to 200 Hz.

Q3: Are there phonetic environments that should be established before you can reliably use low-frequency energy as a cue to distinguish voicing?

Place of Articulation

We will now contrast the distributions of certain acoustic measurements for the alveolar [s z] and palatal [š ž] fricatives. The data set containing these four fricatives, partitioned into alveolar *vs.* palatal, is the **fric_AP** data set.

T7: Using the **fric_AP** data set, examine histogram plots (using `plot_histograms`) for each of the six different measurements and examine their abilities to separate the two classes.

Q4: The ratio of the energy between 2 kHz and 4 kHz and that between 2 kHz and 6 kHz (**MFR Energy**) is a good discriminating measurement. Why?

T8: Generate two-dimensional scatter plots using the `plot_clusters` command for the **fric_AP** data using different pairs of measurements. Include the phone labels in the plot and examine the influence of phonetic context.

Q5: What phonetic context can cause the alveolar fricatives to look like palatal fricatives (in terms of the **MFR Energy** measurement)?

Q6: Based on your investigation of some of the acoustic properties of American English fricatives, as well as other acoustic-phonetic properties discussed in class, briefly describe and justify what acoustic measurements you would use to distinguish the eight fricatives from each other. Specify the context(s) in which each measurement would be used. You can use measurements other than the six included in the MATLAB® data.

Part II: Acoustic Modeling

In this part of the assignment, we will try to model the distributions of the acoustic measurements examined during the first part of the assignment. For the rest of the assignment, we will focus on trying to distinguish between the four fricatives [z š f ð].

T9: Using the **train** data set (and the corresponding **train_index** indices), generate histogram plots (using `plot_histograms`) for each of the six different measurements and examine their ability to separate the four different fricatives.

T10: Generate two-dimensional scatter plots using the `plot_clusters` command for the **train** data. Explore different pairs of measurements and examine their abilities to discriminate between the different classes. For these plots, you do not have to plot the phone labels.

Q7: Based on your observations of the scatter plots, which two measurements are the most correlated? Which two are the most uncorrelated?

T11: We will now model the distributions of the acoustic measurement features using Gaussian and mixture Gaussian distributions. Gaussian and mixture Gaussian models can be trained using the commands **train_gaussian** and **train_mixture**, respectively. Train a single *full-covariance* Gaussian model for the phone [z] (**train_z** data set) using the command:

```
>> [gmean, gvar] = train_gaussian(train_z);
```

The mean vector and the covariance matrix are saved in `gmean` and `gvar`, respectively. We can also get the corresponding *diagonal-covariance* Gaussian model by diagonalizing the covariance matrix (note that the mean vector stays the same):

```
>> dmean = gmean;
>> dvar = diag(diag(gvar));
```

To train a four-component *mixture* diagonal-covariance Gaussian model for the phone [z], use the command:

```
>> [mmean, mvar, mwgt] = train_mixture(train_z,4);
```

The set of mean vectors, variance vectors (only diagonals of the covariance matrices), and mixture weights are saved in `mmean`, `mvar`, and `mwgt`, respectively.

T12: We will now examine how well or poorly the different Gaussian models fit the feature measurements of the training data. To do this, we can compare plots of the measurement distributions with plots of the density estimates of the Gaussian models using the `plot_histograms` and `plot_gaussian` commands. For example, to see the fit of the full-covariance Gaussian model to the first dimension (**MFCC0**) of the data, we can type:

```
>> plot_histograms(train_z,1); hold
>> plot_gaussian(gmean,gvar,1); hold
```

The `hold` command toggles whether to hold on to the current plot, so that another plot can be overlaid, or to draw a new plot. Examine each of the six dimensions to see which dimensions are sufficiently represented by a Gaussian density. To similarly examine the mixture diagonal-covariance Gaussian model, we can use the `plot_mixture` command:

```
>> plot_histograms(train_z,1); hold
>> plot_mixture(mmean,mvar,mwgt,1); hold
```

Examine how well the mixture diagonal-covariance Gaussian model fits each of the six dimensions.

T13: Examination of one-dimensional plots of the data does not allow the observation of any correlation between the different dimensions. To examine how well the models capture correlation information, we can superimpose scatter plots of the training data and two-dimensional "contour plots" of the Gaussian models. Contour plots display lines of equal density. To do this, we can use the `plot_clusters` and `plot_gauss_contour` commands:

```
>> plot_clusters(train_z,2,3); hold;
>> plot_gauss_contour(dmean,dvar,2,3); hold;
```

The above commands will draw a scatter plot of the second and third dimensions of the **train_z** data with the corresponding diagonal-covariance Gaussian density contours superimposed on top. To examine the fit of the full-covariance Gaussian density, we can use:

```
>> plot_clusters(train_z,2,3); hold;
>> plot_gauss_contour(gmean,gvar,2,3); hold;
```

To make a similar plot using the mixture diagonal-covariance Gaussian density:

```
>> plot_clusters(train_z,2,3); hold;
>> plot_mix_contour(mmean,mvar,mwgt,2,3); hold;
```

Examine the diagonal-covariance Gaussian, full-covariance Gaussian, and mixture diagonal-covariance Gaussian models over different pairs of measurements and note how well each model fits the data.

Q8: On the whole, which model (diagonal-covariance Gaussian, full-covariance Gaussian, or mixture diagonal-covariance Gaussian) appears most appropriate for modeling the [*z*] data based on your observations of the different plots? What advantages and shortcomings does each model possess?

T14: Another way to measure how well a model captures information about the training data is to compute the likelihood, or log probability, score of the training data when tested with the trained model. To find the total log probability of the training data when using the diagonal-covariance Gaussian, full-covariance Gaussian, and mixture diagonal-covariance Gaussian models use the following commands:

```
>> sum (log (gaussian_probs (train_z,dmean,dvar)))
>> sum (log (gaussian_probs (train_z,gmean,gvar)))
>> sum (log (mixture_probs (train_z,mmean,mvar,mwgt)))
```

The commands `gaussian_probs` and `mixture_probs` compute the probabilities of each token in the given data using the specified Gaussian or mixture Gaussian model. We then take the log of the probabilities and sum them up over the entire data set to get the total likelihood. This assumes that the individual tokens in the data set are independent of one another.

Q9: By examining the likelihood values of the training data, which model might you expect to perform best when classifying new test data? Under what circumstances might a high training data likelihood not translate into an accurate model when classifying new test data?

T15: Diagonal-covariance Gaussian models assume that the different feature dimensions in the data vectors are independent of one another. As we noticed from the scatter plots above, this is not always the case. This is also evident if we compute the covariance matrices of the training and test data using the command `cov` as follows:

```
>> cov(train)
>> cov(test)
```

Notice that the off-diagonal entries in the matrix are not zero. In these situations, diagonal-covariance Gaussian models may not be the best choice. Two possible solutions to this problem are 1) to use full-covariance Gaussian models and/or 2) to transform the data vectors so that the dimensions are independent. One procedure to perform the latter is called *principal components analysis* or PCA. Generate a new set of training and test vectors by performing PCA using the `pcs` (principal components scaling) command as follows:

```
>> [rtrain, rtest] = pcs(train,test);
```

This command performs principal components analysis on the `train` set data and uses the results to rotate and scale both the `train` and `test` sets, storing them in `rtrain` and `rtest`, respectively. Examine the covariance matrices of the new transformed training and test data:

```
>> cov(rtrain)
>> cov(rtest)
```

Q10: Describe the structure of the covariance matrix of the `rtrain` data. Why isn't the covariance matrix of the `rtest` data the same as that of the `rtrain` data? Is there any reason to prefer using PCA to transform the data versus simply using full-covariance Gaussian models?

Part III: Pattern Classification

In this part of the assignment, we will make use of the modeling techniques that we examined in the previous section to perform pattern classification. In particular, we will examine *maximum likelihood* (ML) classification and *maximum a posteriori* probability (MAP) classification with the different Gaussian models. The two relevant commands are **ml_classify** and **map_classify**. These commands take two sets of data: a training set and a test set. They use the training set to train the parameters of the specified Gaussian model and then use the trained model to perform classification on the test set. For a more detailed description of these commands, enter `help ml_classify` and `help map_classify` at the MATLAB® prompt. Note that we will be using the PCA-rotated and scaled versions of the training (`rtrain`) and test (`rtest`) data sets.

For each of the following tasks, carefully note the classifier's accuracy as the different experiments are conducted. Try to understand the relative merits and shortcomings of each approach.

T16: Perform ML classification on the data using one diagonal-covariance Gaussian density per class by typing the following command:

```
>> ml_classify(rtrain,train_index,rtest,test_index,'diag');
```

This command will train up the specified model using the training data, perform classification on the test data, and return some performance statistics:

```
Percent correct: 86.885

      dh    f   sh    z
 dh  235   10    0   16
 f    11  263    1    5
 sh    0    1  128    5
 z     9   10   68  275
```

The "Percent correct" is simply the accuracy of the model on the test data. The matrix below it is the *confusion matrix*. The rows of the confusion matrix contain the correct phone labels

of the data, while the columns contain the phone labels hypothesized by the classifier. Each entry, $c(i,j)$, in the matrix indicates the number of class i vectors classified as class j. For example, 235 [ð] vectors were correctly classified as [ð] but 10 were misclassified as [f], 16 were misclassified as [z], and none were confused with [š]. Repeat the experiment using the MAP classifier as follows:

```
>> map_classify(rtrain,train_index,rtest,test_index,'diag');
```

Q11: Why does the MAP classifier have a higher accuracy than the ML classifier?

T17: Perform MAP classification using a full-covariance Gaussian density function with the following command:

```
>> map_classify(rtrain,train_index,rtest,test_index,'full');
```

T18: Perform MAP classification with mixture densities using two diagonal-covariance Gaussians per mixture density with the following command:

```
>> map_classify(rtrain,train_index,rtest,test_index,'mix',2);
```

Repeat the above classification experiment several times and note how the performance varies as the mixtures are randomly initialized from different starting points. Then vary the number of Gaussians per mixture to 4, 8, 16, 32, and 64 and note how the performance changes.

Q12: What happens to classification performance as the number of mixture components is increased? Why?

Q13: Which classifier (diagonal-covariance, full-covariance, or mixture diagonal-covariance) has the best performance? Does this match what you anticipated from examining the distributions and likelihoods in the previous section? What reasons might explain the relative performances of each of the classifiers?

T19: By examining the performance of a classifier over varying training set sizes, we can get a sense of how well a classifier's parameters are trained. As the estimates of a classifier's parameters begin to converge (with increasing amounts of training data), it is expected that the classifier's performance on training data vs. test data will also converge. Examine the performance of the full-covariance Gaussian classifier using a training set containing only 77 training vectors (**train1**) with the following commands:

```
>> map_classify(train1,train1_index,rtest,test_index,'full');
>> map_classify(train1,train1_index,train1,train1_index,'full');
```

The first command gives the test set accuracy while second command provides the training set accuracy (i.e., testing on the training set). Repeat the above experiment using different-sized training sets: **train2** (152 vectors), **train3** (302 vectors), **train4** (601 vectors), and **rtrain** (1199 vectors). Note that these training sets have already been properly PCA-rotated and scaled.

Q14: Does the performance of the classifier on train and test data converge as the size of the training set is increased? Do you feel that the full-covariance Gaussian models are sufficiently trained using the full training set or is additional data needed to provide more accurate estimates of the model's parameters?

T20: When only a finite amount of test data is available, it is often difficult to determine whether or not the difference in the observed performance of two different classifiers is statistically significant or not. Other factors can also affect classifier performance; recall the effect of random initialization in the training of the Gaussian mixture models in T18.

To test the statistical significance of the difference in performance between two different classifiers evaluated on the same test set, the McNemar significance test can be used. To perform this test, we need to generate a binary vector for each classifier specifying whether each test vector is correctly or incorrectly identified by that classifier. This can be performed with the following commands:

```
>> cdiag = map_classify(rtrain,train_index,rtest,test_index,'diag');
>> cfull = map_classify(rtrain,train_index,rtest,test_index,'full');
>> cmix  = map_classify(rtrain,train_index,rtest,test_index,'mix',4);
```

The McNemar test for comparing the diagonal-covariance Gaussian classifier with the full-covariance Gaussian classifier can be performed with the following command:

```
>> [S, N] = mcnemar(cdiag,cfull)
```

where `S` is the significance level of the test and `N` is the contingency table used in the test. Repeat the McNemar test to compare the full-covariance Gaussian classifier with the mixture diagonal-covariance Gaussian classifier and to compare the diagonal-covariance Gaussian classifier with the mixture diagonal-covariance Gaussian classifier. Typically the difference in performance of two classifiers is considered statistically significant if the significance level is .005 or smaller.

Q15: What are the McNemar significance levels when comparing the diagonal-covariance Gaussian classifier with the full-covariance Gaussian classifier? The full-covariance Gaussian classifier vs. the mixture diagonal-covariance Gaussian classifier? The diagonal-covariance Gaussian classifier vs. the mixture diagonal-covariance Gaussian classifier? Considering these values, how would you qualitatively describe the relative performances of the three different classifiers?