Laboratory # 9

Vowel Lab III: Vowel Classification

9.1 Introduction

Classification is ubiquitous. The ability to recognize and categorize things based on their features is fundamental to human cognition; a large part of our ability to understand and deal with the world around us is a result of our ability to classify things. Similarly, we want to design systems that are capable of recognition and categorization. We want vending machines to be able to recognize the bills inserted into the bill changer. We want internet search engines to classify web pages based on their relevance to our query. We want computers that can recognize and classify speech properly so that we can interact with them naturally. We want medical systems that can classify unusual regions of an x-ray as cancerous or benign. We want high speed digital communication modems that can determine the sequence of, say, 64-ary signals that were transmitted.

There is a vast array of applications for classification. In this lab, we will build upon the foundation that we developed in Lab 8 to actually perform classification of vowel sounds. We will explore a couple of ways of measuring the performance of a *classifier* (i.e., a system that performs classification), and we will relate these performance measures back to the ones that we introduced in Lab 8 for characterizing different feature sets.

9.2 Background

As illustrated by the block diagram in Figure 9.1, a classifier has two main subsystems: a *feature calculator* and a *decision maker*. The feature calculator takes the instance to be classified and outputs a feature vector. For example, in the case of vowel classification, the input will be a segment of recorded speech, and the output could be any of the three types of feature vectors considered in Lab 8: formants, AR feature vectors, or spectral feature vectors.

The decision maker takes the feature vector as its input and produces a class name or label, according to some prespecified *decision rule*. For example, in Lab 8 we suggested that there could be a representative feature vector for each class, and the decision rule could decide class C if the input feature vector is closer[1] to the representative of class C than to any other class. Though we will consider variations of this, all the decision rules in this lab will be based on comparing the distances between the input feature vector and a set of class representatives. Accordingly, Figure 9.1 shows that the decision maker can be subdivided into a *distance calculator*, which computes the distances between the input feature vector

[1] in Euclidean distance

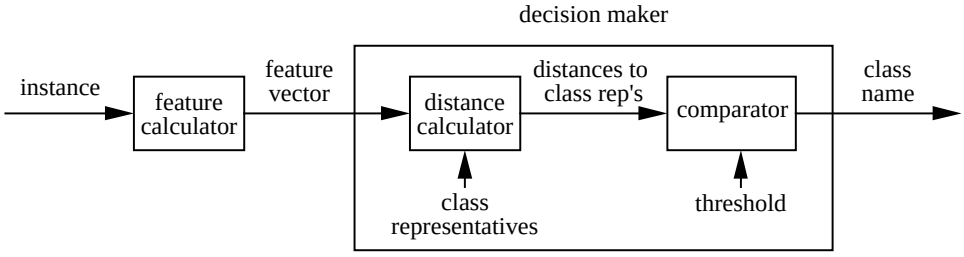

Figure 9.1: Block diagram of a classifier.

and the representatives, and a *comparator*, which compares the distances and produces a classification using some rule.

The representative feature vectors are generally determined during the design phase, and Figure 9.1 shows them as *side* inputs to the distance calculator. As suggested in Lab 8, we will take the representative from each class to be the mean of the feature vectors from a sizeable number of instances that are known to be from that class. Such instances or their corresponding feature vectors are said to form a *training set*, since they are used to train, i.e. design, the classifier.

In some cases, the comparator will have an additional parameter that can be adjusted during the design. Accordingly, Figure 9.1 shows a parameter called a *threshold*, which is introduced in the next subsection.

Generally speaking, the goodness of a classifier is determined by how frequently it makes correct and incorrect decisions. However, as we discuss in the next section, we often distinguish between different types of correct and incorrect decisions.

9.2.1 Two-Class Classification

In this subsection we examine in detail the specific case of a two-class classifier. Suppose we are trying to determine if an instance belongs to Class A, but we expect to encounter instances from both Class A and some other class, Class B[2]. That is, Class A is the *class of interest*. Further, suppose we have used a set of training data from each class to compute a pair of mean feature vectors, which we use as class representatives. To classify a particular instance, we first calculate the feature vector for that instance. Then, we calculate the distances between this resulting feature vector and the two mean feature vectors. Intuitively, it makes sense to *decide* the class that corresponds to the smaller of the two distances. That is, we classify the instance as a member of the class whose mean feature vector is closest to the given feature vector. If the distances are denoted d_a and d_b, respectively, we can write this decision rule as

$$d_a \underset{B}{\overset{A}{\lessgtr}} d_b \tag{9.1}$$

That is, if d_a is less than d_b, we select A; otherwise, we select B. Note that this decision rule also works if we find class representatives in some other fashion. We will call this simple decision rule the *nearest-representative decision rule*.

What are the possible results of this classification? It is possible that the new instance actually belongs to Class A and the classifier properly decides Class A. The system has properly classified an instance as matching the class of interest, so this decision is called a *hit*. When the instance belongs to Class A, but the system fails to recognize this, we call this decision a *miss*. Alternatively, it is possible that the instance is not a member of Class A. If the classifier correctly recognizes that the instance is not a member of Class A (i.e., it decides Class B), the decision is called a *correct rejection*. If the classifier does *not* reject

[2]Class B could simply be anything that isn't in Class A.

		Truth: Class A	Truth: Not Class A
Decision	Class A	Hit	∗False Alarm
	Not Class A	∗Miss	Correct Rejection

Table 9.1: Types of decisions based on the truth and the decision made. Errors are marked with an ∗.

the instance, though, the decision is called a *false alarm*. Table 9.1 summarizes these types of decisions.

It is important to note the relationship between these various types of decisions. If an instance is a member of Class A (the class of interest), the resulting decision must be either a hit or a miss. Thus, if we test a classifier on ten instances from the class of interest and eight of them are classified correctly, we know that the classifier has gotten eight hits and two misses. The fraction of test instances *from the class of interest* that yield hits is known as the *hit rate*, r_h, while the fraction of tests that yield misses is known as the *miss rate*, r_m. Note that $r_m = 1 - r_h$. Similarly, if the instance is not a member of the class of interest, the decision must either be a correct rejection or a false alarm. We define the *false alarm rate*, r_{fa}, as the fraction of test instances *not* from the class of interest that yield false alarms; the fraction of these instances that yield correct rejections is the *rejection rate*, r_r. As with the hit rate and miss rate, $r_r = 1 - r_{fa}$.

Because of the relationships between r_h and r_m it suffices to consider only one of them. Similarly, it suffices to consider only one of r_r and r_{fa}. From now on, we will follow the usual convention by focusing only on the hit and false alarm rates, r_h and r_{fa}. Sometimes we may want to achieve a particular value for one of these rates. The next section provides an example of when this might occur and suggests a method that allows the classifier to be tuned to adjust these rates.

9.2.2 Looking at the Tradeoffs: Cancer Diagnosis

Consider the following classification problem. We have built a system that scans an x-ray for potentially cancerous legions and calculates a certain feature vector for each lesion. We have also collected feature vectors from a large number of lesions; some of these lesions are known to be cancerous, while the rest are known to be benign. From these, we calculate the mean feature vectors for these two classes to be used as class representatives. Now suppose our system is given an x-ray that has one potentially cancerous legion. We calculate the feature vector for this lesion, and calculate the distance from this feature vector to the mean feature vectors for cancerous and benign lesions. How do we make a decision in this case?

Of course, we could simply use the nearest-representative decision rule introduced above. However, we've neglected to consider the consequences of our decision. On the one hand, suppose the decision is a false alarm; that is, we label the lesion cancerous, but it is actually benign. In this case, we subject our patient to a time consuming (and expensive) procedure to determine if the lesion is in fact cancerous. Maybe we should be a little more conservative, and tend to label fewer things as cancerous. On the other hand, suppose we "miss" and label a lesion that is truly cancerous as benign. Our patient may not be diagnosed until the cancer has progressed significantly; this is a potentially life-threatening situation. So maybe we should be a little more proactive and classify more things as cancerous.

How can we tune our classifier to take these preferences into account? A simple way is to use a *decision threshold* τ. Specifically, let τ be a nonnegative number and consider the decision rule that decides Class A if the distance of the instance feature vector to the mean feature vector of class A is less than τ times the distance to the mean feature vector of Class

B. That is, the decision rule is

$$\frac{d_a}{d_b} \underset{B}{\overset{A}{\lessgtr}} \tau \tag{9.2}$$

For example, if $\tau = 2$, the rule favors Class A, because it decides Class A even if the distance to the Class A mean feature vector is, say, 1.9 times larger than the distance to the Class B mean feature vector. This increases the hit rate r_h at the expense of also increasing the false alarm rate r_{fa}. On the other hand, if $\tau < 1$, the decision rule favors Class B, which decreases the hit and false alarm rates.

As introduced, this decision threshold is somewhat awkward. To favor Class A (that is, to choose Class A more often), τ must be in the range of $(1, \infty)$, but to favor Class B, τ must be in the range of $(0, 1)$. The two types of thresholds are related by a reciprocal operation, which is not intuitive. We can produce a simpler decision threshold, τ', by taking the logarithm of both sides, as in

$$\log\left(\frac{d_a}{d_b}\right) \underset{B}{\overset{A}{\lessgtr}} \log(\tau) = \tau' \tag{9.3}$$

Now, to favor Class A we simply use a positive decision threshold; to favor Class B, we use a negative decision threshold. Further a decision threshold of $\tau' = 2$ favors Class A as much as a threshold of $\tau' = -2$ favors Class B, yielding a much more intuitive relationship. The nearest-representative rule, as given in equation (9.1), is equivalent to $\tau' = 0$.

9.2.3 Tuning the Classifier: ROC Curves

How do we examine the effect of the threshold on classifier performance? Suppose we set a particular threshold and measure the system's false alarm rate and its hit rate[3]. Then, we adjust the threshold and measure these rates again. If we continue this for a large number of thresholds, we will have obtained enough data to characterize the range of potential classifier performance. It is traditional to plot the hit rate as a function of false alarm rate. The resulting plot is known as an *ROC curve*[4]. The different points along the ROC curve (called *operating points*) correspond to different decision thresholds.

Figure 9.2 shows several possible ROC curves. Different curves correspond to different classification tasks, e.g. classifying lesions from x-rays vs. MRI scans, or classifying different types of bills as valid or invalid. They may also correspond to different choices of features upon which to base the distances. However, in all cases the two endpoints of the curve (at $(0, 0)$ and $(1, 1)$) correspond to the same trivial decisions: at (0,0), nothing is classified as belonging to the class of interest; similarly, at (1,1), everything is classified as belonging to the class of interest.

Consider the asterisk on the upper (red) curve. For this threshold, our hit rate is 0.68; that is, the classifier correctly recognizes 68% of the members of the class of interest[5]. At this operating point, the classifier also has a false alarm rate of 0.2. By adjusting the threshold, we can move along this curve. We can reduce the false alarm rate by moving to the left on the curve, but we suffer a corresponding decrease in hit rate. Similarly, we can increase the hit rate, but only by accepting an increase in the false alarm rate as well.

In an ideal world, the hit rate would be 1 with a false alarm rate of 0. Thus, ROC curves which are closer to the upper left corner of the plot are desirable. The straight dotted line on Figure 9.2 is the ROC curve for a *random decision rule*, which randomly decides Class A or B without regard to the instance or its feature vector. For example, it might decide Class A with probability 0.8 and Class B with probability 0.2. In this case, it has a hit

[3] Recall that we can calculate the rejection rate and miss rate from these values

[4] "ROC" stands for *receiver operating characteristic*, a term which comes from the field of radar detection.

[5] Equivalently, we can note that the miss rate is 0.32.

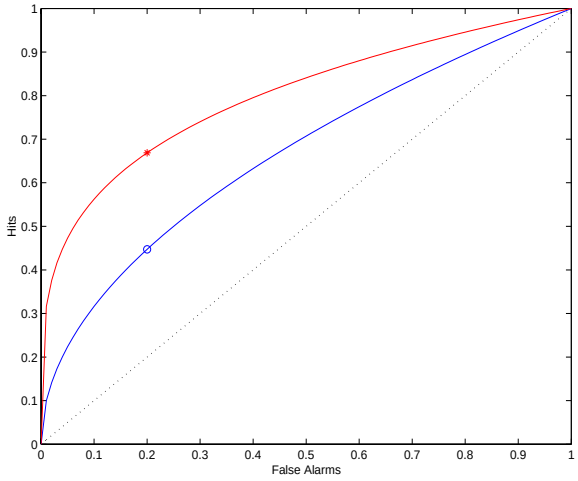

Figure 9.2: An example ROC curve (solid). The dotted line indicates the performance of a random classifier.

rate of 0.8 and a false alarm rate of 0.8. Indeed, it always has the same hit and false alarm rates; this explains why the ROC curve is a straight line from (0,0) to (1,1). We conclude that any *reasonable* classifier should have an ROC curve lying above the straight line from corner to corner, because any classifier whose ROC curve falls below this diagonal is doing worse than chance[6]!

In summary, we view the ROC curve as a measure of the difficulty of the classification task or the goodness of the chosen set of features. A difficult task or a poor set of features results in an ROC curve that is close to the corner-to-corner straight line. An ROC curve closer to the upper left corner indicates an easier task and/or a good choice of features.

There are two important things to note. First, the ROC curve itself does not indicate what threshold should be used to achieve a particular hit or false alarm rate. Instead, the ROC curve gives us an overall picture of the potential range of classifier performance as threshold varies. To find an appropriate threshold, we need to work backwards and find the threshold used to achieve a particular hit rate or false alarm rate. Second, the ROC curve is only meaningful for a two-class classifier. However, one could use it with multiple-class classifiers (like our five-vowel classifier) by considering the binary decision "class n or not class n."

9.2.4 Multi-class Classifiers

In this lab, we will be developing a five-class vowel classifier which uses the nearest-representative decision rule. Note that it would be possible to modify this rule by assigning thresholds to each pair of classes, thus permitting certain classes to be favored over others. Aside from the difficulty of deciding which classes should be favored, there is the additional difficulty associated with adjusting such thresholds systematically. ROC curves are not meaningful for multiple-class classifiers, primarily because we cannot characterize the performance of such a classifier along only two dimensions. For the vowel classifier examined here, it turns out that there is no reason to favor one type of error over any other; they are all equally undesirable. There is no particular reason to adjust such thresholds. Thus, we will simply use the nearest-representative decision rule for multi-class classification.

[6] If we recognize this, we can sometimes reverse the classifier's decisions to yield a classifier that performs better than chance.

9.2.5 Training Classifiers

When designing classifiers and testing their performance, it is important to note that classifiers generally perform better on the training data used in their design than on new data of the same general type. Thus, to objectively assess the performance of a classifier, one must test it on a different data set, usually called a *test set*. To see why, consider the extreme case in which the training data contains just one feature vector for each class, which becomes the mean feature vector for its class. In this case, the resulting classifier will perfectly classify every feature vector in the training set. However, it may not do very well at all when classifying other data. In more realistic cases where the training data has quite a few instances of each class, the performance of the classifier on the training data will usually be somewhat (but usually not significantly) better than on test data. Nevertheless, it is widely accepted that testing a classifier on independent data is good practice. Thus, when a certain amount of data is available for design, it is usually divided into two sets — one for training, the other for testing.

9.2.6 Confusion matrices

One good way to characterize the performance of a multi-class classifier is with a *confusion matrix* K, which simply measures how often members of one class were confused with members of another class. Specifically, when the classifier recognizes N classes, then the confusion matrix $K = [K_{i,j}]$ is an $N \times N$ matrix, whose element $K_{i,j}$ in the i^{th} row and j^{th} column is the fraction of those times that class j occurs but the classifier produces class i. That is,

$$K_{i,j} = \frac{\#\text{ of class } j \text{ instances classified as } i}{\#\text{ of class } j \text{ instances}} \tag{9.4}$$

For example, the following is confusion matrix for a hypothetical 4-vowel classifier:

$$K = \begin{bmatrix} .9 & .03 & .01 & .02 \\ .03 & .95 & .01 & .03 \\ .05 & .01 & .96 & .1 \\ .02 & .01 & .02 & .85 \end{bmatrix} \tag{9.5}$$

The diagonal elements of the confusion matrix show what fraction of instances from the appropriate class were correctly classified. Alternatively, the diagonal element $K_{n,n}$ shows the hit rate for the n^{th} class as the class of interest. The off-diagonal elements indicate what percentage of each class was improperly classified as one of the other classes. Alternatively, the off-diagonal element $K_{n,m}$ is a sort of "false alarm rate," with the m^{th} class as the class of interest. All of this means that we prefer a classifier with a confusion matrix with large values on the diagonals and small values off-diagonal. The confusion matrix for a perfect classifier will be an identity matrix (i.e., ones on the diagonals, zeros elsewhere).

How does the confusion matrix K relate to the distance histogram plots and cluster matrices C that we calculated in Lab 8? Remember that a "good" cluster matrix will have small values along the diagonals and large values for off-diagonal elements. This is the opposite of the trend for a "good" confusion matrix. Because of the way that we've defined these matrices, element $K_{i,j}$ should roughly correspond to the inverse of element $C_{i,j}$. You'll explore this connection in the lab assignment. Also, remember that the cluster matrix is basically a summary of the various distance histogram plots; just as the n^{th} column of C reflects the distance histogram plot with the n^{th} class as the class of interest, the same column of K also has some connection to this same plot.

9.2.7 MATLAB Refresher

To determine if elements in a vector or matrix, `x`, are equal to a particular value (say, 12), use the command

```
cmp = (x == 12);
```

The result, `cmp`, is a vector or matrix with the same size as `x` that contains only ones and zeros.

To find the indices of all elements in a vector or matrix, `x`, that equal a particular value (say, 42), use the command

```
indices = find(x == 42);
```

The result, `indices` is a vector of indices, one for each matching element in `x`, indicating the locations of matching values.

To count the number of elements in a vector, `x`, that are equal to some value (say, 3), use the command

```
count = length(find(x == 3));
```

To calculate the fraction of elements in a vector, `x`, that are equal to some value (say, 143), use the command

```
frac = length(find(x == 143))/length(x);
```

9.2.8 MATLAB Functions for this Lab

The following functions are provided for your use in this lab.

1. `M = train_classifier(fm_train_1, fm_train_2, ... )` — This function trains a classifier by computing a class representative for each of its classes.

   As input, the function accepts any number of feature matrices (`fm_train_1`, `fm_train_2`, etc.), one for each class to be recognized. A *feature matrix* is a matrix whose rows are feature vectors. The feature vectors in the matrix `fm_train_n` are the training data for the n^{th} class.

   The function returns a *classifier matrix* `M`, which contains the class representatives as its rows. In particular, as a representative of the n^{th} class, its n^{th} row contains the mean of the feature vectors in `fm_train_n`.

2. `labels = two_class_test(M, fm_test, thresh)` — This function tests a two-class classifier on a set of testing data, with an optional threshold.

   As input, the function accepts a classifier matrix `M`, as produced for example by `train_classifier`, and a test feature matrix, `fm_test`. A third optional parameter, `thresh`, specifies a threshold in the range $(-\infty, \infty)$; this threshold is of the form given for τ' in equation (9.3). If unspecified, the threshold defaults to 0, yielding the simple nearest-representative rule.

   The function returns a vector of integer class labels `labels`, with one label for each instance in `fm_test` in the the original training order (see `train_classifier`).

3. `[hit_rate, false_alarm_rate, thresholds] = two_class_calc_roc(M, fm1, fm2)` — This function (which you will need to complete in the lab assignment) tests a two-class classifier for a number of different thresholds and returns the associated hit rates and false alarm rates.

   As input, the function accepts a classifier matrix, `M`, as produced for example by `train_classifier`, a testing set feature matrix from the first class, `fm1`, and a testing set feature matrix from the second class, `fm2`.

   The function returns three vectors, all of which will have the same length. `hit_rate(n)` and `false_alarm_rate(n)` contain the respective rates for the threshold given by `thresholds(n)`. Note that `two_class_calc_roc` calls `two_class_test`, so the thresholds are again of the form given for τ' in equation (9.3).

4. `labels = test_classifier(M, fm_test)` — This function tests a classifier on a set of testing data. The classifier can have as many classes as desired.

   As input, the function accepts a classifier matrix, `M`, as produced for example by `train_classifier`, and a test feature matrix, `fm_test`.

   The function returns a vector of integer labels — one for each instance in `fm_test`, in the original training order (see `train_classifier`).

5. `K = confusion_matrix(M, fm_test1, fm_test2, ... )` — This function computes the confusion matrix for a multi-class classifier on test data.

   As input, the function accepts a classifier matrix, `M`, and one test feature matrix (`fm_test1`, `fm_test2`, etc.) for each class.

   The function returns the calculated confusion matrix.

6. `C = cluster_matrix_tt(M, fm_test1, fm_test2, ... )` — This function computes the cluster matrix for a multi-class classifier[7].

   As input, the function accepts a classifier matrix, `M`, and one test feature matrix (`fm_test1`, `fm_test2`, etc.) for each class in `Class`. Note that this is the same set of input parameters as `confusion_matrix`.

   The function returns the calculated cluster matrix.

9.3 Demonstrations in the lab session

Intersperse the Matlab demos listed in the last item with the appropriate items listed below.

1. Two-class classification
2. "Tuning" the classifier and using a threshold
3. ROC Curves
4. Training versus testing
5. Multi-class Classification
6. Classification in MATLAB

9.4 Laboratory assignment

Download the file `lab9_data.mat`. This file contains a large number of feature matrices. Specifically, for each of five vowels and each of the three types of features considered in Lab 8, there are two feature matrices, one to be used for training and the other to be used for training. Note that there is nothing special about the instances in the training matrices versus the testing matrices. All of the instances come from the same source; they have simply been divided into training and testing sets for your convenience. Table 9.4 lists the names of the feature matrices in `lab9_data.mat` to be used for each purpose.

1. Download the files `train_classifier.m` and `two_class.m`. Train a classifier on the AR feature matrices given in `oh_ar_trn` and `ae_ar_trn` using the command

   ```
   M = train_classifier(oh_ar_trn,ae_ar_trn);
   ```

[7] Note that this function is a modification of the function `cluster_matrix`, which was used in Lab #8. In this function, the class mean vectors are calculated separately using `train_classifier`. That is, this function allows "training" on a different data set than is tested on.

Vowel	Feature Types		
	Formant	AR	Spectral
ee	`ee_f_trn`	`ee_ar_trn`	`ee_sp_trn`
	`ee_f_tst`	`ee_ar_tst`	`ee_sp_tst`
ae	`ae_f_trn`	`ae_ar_trn`	`ae_sp_trn`
	`ae_f_tst`	`ae_ar_tst`	`ae_sp_tst`
ah	`ah_f_trn`	`ah_ar_trn`	`ah_sp_trn`
	`ah_f_tst`	`ah_ar_tst`	`ah_sp_tst`
oh	`oh_f_trn`	`oh_ar_trn`	`oh_sp_trn`
	`oh_f_tst`	`oh_ar_tst`	`oh_sp_tst`
oo	`oo_f_trn`	`oo_ar_trn`	`oo_sp_trn`
	`oo_f_tst`	`oo_ar_tst`	`oo_sp_tst`

Table 9.2: The feature matrices to be used for classification. Training sets end with "trn," while testing sets end with "tst."

Now, test the classifier on the matrices `oh_ar_tst` and `ae_ar_tst` with the nearest-representative rule using the commands

```
labels1 = two_class_test(M,oh_ar_tst);
labels2 = two_class_test(M,ae_ar_tst);
```

(a) Suppose that "oh" is our class of interest, so we consider a correct classification of an "oh" to be a hit and a correct classification of an "ae" to be a correct rejection.

- [4] What is the hit rate for the classifier you've just trained?[8]
- [4] What is the false alarm rate for the classifier you've just trained?

(b) Repeat the testing you performed in Problem 1a using a threshold of $\tau' = -0.5$.

- [4] What is the hit rate using this threshold?
- [4] What is the false alarm rate using this threshold?
- [4] Compare these results to the results you calculated in Problem 1a. Which class does this threshold favor, compared to a threshold of 0?

(c) Find a threshold for the classifier used in Problem 1a that yields a hit rate of 0.7. (Hint: Consider plotting the hit rate as a function of the threshold.)

- [4] What is the threshold that you found?
- [3] What is the associated false alarm rate?

2. Now we'd like to compute some ROC curves to show the range of potential performance for the "oh" classifiers. Download the file `two_class_calc_roc.m`.

(a) `two_class_calc_roc` is not quite complete; there are several commands (each indicated with a `?`) that you need to fill in.

- [10] Complete this function and include the code in your report.

(b) Use `two_class_calc_roc` to calculate hit and false alarm rates for the classifier calculated in Problem 1. Plot the resulting ROC curve. (Hint: Remember that the false alarm rate is plotted along the x-axis, and the hit rate is plotted along the y-axis.)

- [4] Include the resulting ROC curve in your report.
- [3] From this curve, determine the maximum hit rate that can be achieved with a false alarm rate of 0.

[8] Make sure you include the code you use to calculate these values.

(c) Use `train_classifier` to train three new two-class classifiers based on the AR coefficient features using the commands

```
M2 = train_classifier(oh_ar_trn,ee_ar_trn);
M3 = train_classifier(oh_ar_trn,ah_ar_trn);
M4 = train_classifier(oh_ar_trn,oo_ar_trn);
```

Using the appropriate testing data, compute a ROC curve for each of these classifiers. (Hint: Make sure you use the appropriate *testing* data for this).

- [6] Plot these three ROC cuves and the one from Problem 2b on the same figure.
- [3] From these curves, determine which vowel is most likely to be confused with "oh."
- [3] Which vowel is least likely to be confused with "oh"?

3. Now, let's move to multi-class classification. Download the files `test_classifier.m` and `confusion_matrix.m`. Train a five-vowel classifier on the AR feature matrices using the following command

```
M5 = train_classifier(oh_ar_trn,ae_ar_trn,ee_ar_trn,ah_ar_trn,oo_ar_trn);
```

Note the order of the classes.

(a) Compute the confusion matrix for this five-class classifier with the command

```
K = confusion_matrix(M5,oh_ar_tst,ae_ar_tst,ee_ar_tst,ah_ar_tst,oo_ar_tst);
```

- [3] Include this confusion matrix in your report.
- [3] Which class is correctly classified the least often?
- [3] Which class is correctly classified the most often?

(b) Use `confusion_matrix` to compute the confusion matrix for the case where we test on the training data rather than on a set of novel instances.

- [3] Include this confusion matrix in your report.
- [3] Compare this confusion matrix to the one calculated in Problem 3a. Which shows better performance?

(c) Let's now explore the connection between the actual classifier performance (as indicated by the confusion matrix) and the performance one might expect based on the the cluster matrix (see Lab 8). Download `cluster_matrix_tt` and calculate the cluster matrix using the testing data. (Hint: Use the same input parameters that you used for `confusion_matrix` in Problem 3a).

- [3] Include the cluster matrix in your report.
- [3] Compare the cluster matrix with the confusion matrix calculated in Problem 3a. Do the two matrices indicate the same relative performance for each class?

4. Finally, let's look at the performance of the other two feature sets.

(a) Train a five-vowel classifier using the formant features, making sure to use the same class order given in Problem 3. Calculate the confusion matrix for this classifier.

- [4] Include this confusion matrix in your report.

(b) Train a five-vowel classifier using the spectral features, again making sure to use the same class order given in Problem 3. Calculate the confusion matrix for this classifier.

- [4] Include this confusion matrix in your report.

(c) Now, compare these two confusion matrices with the one calculated in Problem 3a.

- [3] From these three confusion matrices, which feature set provides the best classification performance?
- [3] Which feature set provides the worst classification performance?
- [3] Does this match the conclusions you drew in Lab 8 based on the distance histogram plots and cluster matrices?

(d) Finally, let's look at what these confusion matrices say about the various classes.

- [3] Do certain vowel classes have uniformly better or worse performance than others? If so, which ones?
- [3] Are there vowels that are consistently confused with some other vowel, regardless of which feature set is used? If so, which ones? Which vowels are they confused with?