Laboratory # 8

Classification and Vowel Recognition

8.1 Introduction

The ability to recognize and categorize things 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. This task, which is generally known as *classification*, is important enough that we often 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 consider a popular application of classification: speech recognition. In particular, we will focus on a simplified version of speech recognition, namely, *vowel classification*. That is, we will experiment with systems that classify a short signal segment, corresponding to a spoken vowel, as either an "ah", or an "ee", or an "oh", etc.. (We won't deal with how one determines that a given segment corresponds to a vowel.) In the process, we will develop some of the basic ideas behind automatic classification.

One of these basic ideas is that an item to be classified is called an *instance*. For example, if each of 50 short segments of speech must be individually classified, then each segment is considered to be one instance. A second basic idea is that there is a finite set of prespecified *classes* to which instances may belong. The goal of a *classifier system* (or simply a *classifier*) is to determine the class to which a presented instance belongs. A third basic idea is that to simplify the process, the classification of a given instance is based on a set of *feature values*. This set is a relatively small list of numbers that, to an appropriate degree, describe the given instance. For example, the short segment of speech might contain 1000 samples, but we will see that vowel classification can be based on feature set with as few as two components. A fourth basic idea is that classification is often performed by comparing the feature values for an instance to be classified with sets of feature values that are *representative* of each class.

The output of the classifier will be the class whose representative feature values are most similar, in some appropriate sense, to the feature values of the instance to be classified.

8.1.1 "The Question"

- What is the general framework for performing automatic classification?
- How can we recognize and classify vowels in a speech recognition system?

8.2 Background

8.2.1 An Introduction to Classification

You may recall Lab 7, in which we developed a system for decoding DTMF signals into the sequence of key-presses that produced the original signal. Our DTMF decoder was actually performing classification on each segment of the DTMF signal. Classification is a process in which we examine *instances* of some thing (like an object, a number, or a signal) and try to determine which of a number of groups, or *classes*, that instance belongs to. We can think of this as a labeling process. In our DTMF decoder, for example, we looked at a given segment of the signal and labeled it with a number corresponding to an appropriate key press.

Generally, classification is a two-stage process. Figure 8.1 shows a block diagram of a classifier system. First, we need some information about the instance that we are considering. This information is traditionally referred to as a set of *features*. If we are classifying people, for instance, we might use height, weight, or hair color as features. If we are classifying signals, we might use power, the output of some filter, or the energy in a certain spectral band as features. So that we can deal with our features easily, we generally like to have a set of measurable features to which we can assign numerical *feature values*. When we are using more than one feature to describe an instance, we typically place all of the feature values into a *feature vector*, $\mathbf{f} = (f_1, f_2, \ldots, f_N)$. N is the number of elements in the feature vector and is called the *dimension* of the feature vector. A feature vector is calculated for each instance we wish to classify by measuring the appropriate aspects of that instance. As shown in Figure 8.1, the first block is the "feature calculator," which takes an instance (of a signal, for instance) and produces the set of numerical feature values. For our DTMF decoder, our features were the spectral strength of a given segment of the signal at each DTMF frequency. That is, the feature calculator produced a seven-element feature vector, one for each DTMF frequency.

The second stage of classification, the "feature classifier," uses the feature vectors to decide which class a feature vector belongs to. Generally, we make this decision by comparing the feature vector for an instance to each member of a set of *representative feature vectors*, one for each class under consideration. The idea is that the decision-maker labels the instance as the class that has the most similar representative feature vector. We will discuss the specifics of the feature classifier after we have presented a classification example.[1]

Before we continue, we should note the relationship between what we previously called "detection" and what we now call "classification". Detection generally refers to binary "signal present" or "signal not present" decisions. For instance in Lab 1, we used to energy

[1]The classifier for the DTMF decoder can be viewed as implicitly operating in this fashion. It is an interesting exercise to find the representative feature vectors implicitly used.

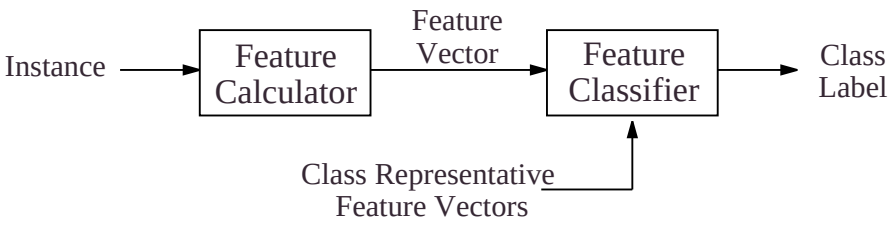

Figure 8.1: Block diagram of a general classifier system.

to decide whether a signal was present or not, and Lab 2 we used correlation to make such decisions. As such, detection, is generally considered to be a special case of the more general notion of classification, which refers to decisions among two or more classes. However, this usage is not universal. For example, "detection" is sometimes used to describe a system that decides which of 64 potential signals was transmitted to a modem, each representing a distinct pattern of 6 bits. This lab assignment also generalizes the idea, used in Labs 1 and 2, that decisions are made on a single number or feature. However, as noted before, Lab 7 also used such a generalization.

8.2.2 A classification example

The easiest way to get a feel for classification problems is to consider an example. Suppose that we have a large number of flowering plants in our garden, each of which belongs to one of two different types, A and B. We know which plant belongs to each type, but they all look very similar. Now, we may know which of our plants belong to which type, but we would also like to be able to classify new plants as either Type A or Type B as we expand our garden. To do this, we will *design* a classifier for these plants. In this design, the plants in our garden with known type will form our *design set* (or *training set*). They will be used for designing the classifier.

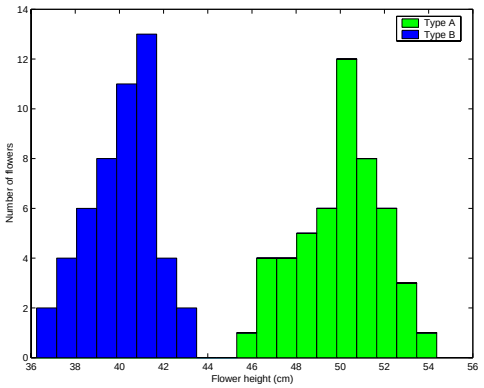

Figure 8.2: A simple example where one feature (plant height) is sufficient to perform classification. This histogram shows how many plants have a given height.

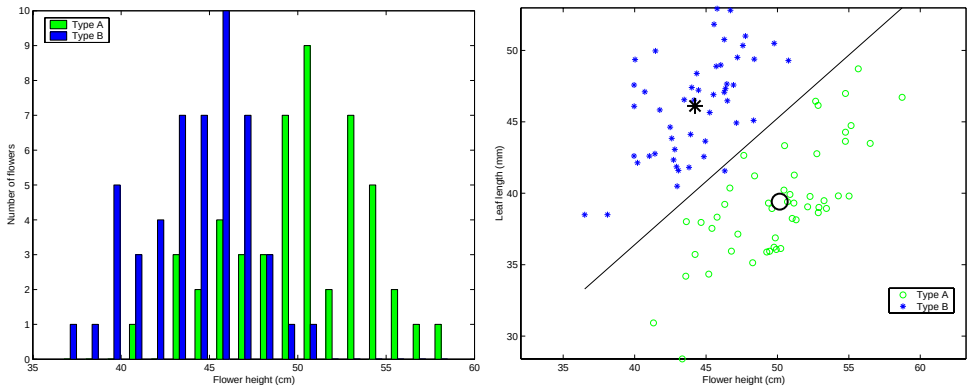

Figure 8.3: An example where a histogram of one feature is *not* sufficient to perform perfect classification (left), but a scatter plot of two features shows a clear separation between the two classes (right).

If we happen to know that Type A plants tend to be taller than Type B plants, this suggests that we might be able to use the plant's height as a feature for classification. Suppose we measure the heights of all of the plants in our garden (our design set) and then plot a histogram of this data. We might see something like Figure 8.2. This is an unusually good case. Notice the two clusters on this histogram. The type A plants form a cluster with heights centered around a mean (i.e. average) of 50 centimeters, and the type B plants form a cluster with heights centered around a mean of 40 centimeters. Most importantly, the two clusters do not overlap. This suggests that classification can indeed be based on plant height.

How do we use this information to classify a new plant (i.e., a new instance)? Intuitively, if the new plant's height is closer to the Type A mean of 50 cm than to the Type B mean of 40 cm, we should classify the plant as type A rather than type B. In this case, we can use a simple threshold test. If a new plant's height is greater than 45 cm (which is halfway between two mean feature values), we classify the new plant as Type A. Conversely, if it's height is less than 45 cm we classify it as Type B. In other words, for each class, we use the mean feature value as the class representative, and we compare the feature value of a new instance to be classified (it's height) to the two means and *decide* the class whose representative feature value (its mean) is closest to the feature value of the given instance.

Figure 8.3 (left) shows a histogram of plant heights in a more troublesome scenario. In this case, Type A plants still tend to be taller than Type B plants, but there are a significant number of plants that we will confuse (that is, misclassify) if we decide exclusively using this one feature. Though we will typically need to deal with some classification error, we can often reduce it by adding more features. Suppose we measure not only the height of the plant but also the average length of its leaves. Now, instead of a histogram, we can look at the training set of features using a *scatter plot*, in which we plot a point for each feature vector in our training set. We we put one of the two features along each of the plot's axes. For example, a scatter plot for the two features just mentioned for each plant is shown in Figure 8.3 (right). Here we again see two distinct clusters, which suggests that classification may be done well based on these two features together.[2]

[2]Note that classification could not be done well using either feature by itself. That is, by itself, neither

How do we design a classifier for this case? We cannot simply use a threshold on one of the features. Instead, we will use a more general decision rule, which is based on mean feature vectors and distances between an instance and the mean feature vectors. First, let us consider the two features as a two-dimensional vector $\mathbf{f} = (f_1, f_2)$. Thus, if a plant is 52 cm tall and has leaves with average length of 44 mm, our feature vector is $\mathbf{f} = (52, 44)$. Now, given the feature vectors from each plant in our design set of one type, we want to calculate a *mean feature vector* for plants of that type. Since the mean feature vector indicates the central tendency of each feature in a class, we use it as a *representative* of the entire class. To calculate a mean feature vector in this case, we first take the mean, m_1, of all of the plant heights for plants of one type. Then we take the mean, m_2, of all of the leaf lengths for plants of the same type. The mean feature vector is then $\overline{\mathbf{f}} = (m_1, m_2)$. Note that this is the general procedure for calculating the mean of a set of vectors, regardless of the vector's dimension. On the scatter plot in Figure 8.3, we've plotted the locations of mean feature vectors with large symbols.

As with the one-feature case, we will classify new instances based on how close they are to each of the mean feature vectors. To do this, we still need to know how to calculate distances between two feature vectors. For simplicity, we will calculate distances using the Euclidean distance metric[3]. The Euclidean distance between two vectors is simply the straight-line distance between their corresponding points on a scatter plot like that in Figure 8.3. To calculate the distance, d, between two feature vectors (f_1, f_2) and (m_1, m_2), we simply use the formula

$$d = \sqrt{(f_1 - m_1)^2 + (f_2 - m_2)^2} \tag{8.1}$$

Euclidean distance generalizes to any number of dimensions; the general formula can be found later in equation (8.2). Note that the Euclidean distance is essentially the RMS difference (i.e., RMS "error") between two vectors[4], which we have used repeatedly throughout this course. Here, though, we refer to the computation as "Euclidean distance", rather than RMS difference, to motivate a geometric interpretation of classification.

Now that we have designed a classifier for this case, we can finally consider the classification of a new instance. To classify a new instance, we first calculate the distances between that instance's feature vector and the mean feature vectors of each class. Then, we simply classify the instance as a member of the class for which the distance is smallest. Consider what this means in terms of the scatter plot. Given a new instance, we can plot it's feature vector on the scatter plot. Then, we classify based on the nearest mean feature vector. For a two-class case such as that shown in Figure 8.3, there exists some set of points that are equally far from both mean feature vectors. These points form a *decision line* that separates the plane into two halves. We can then classify based on the half of the plane on which a feature vector falls. For example, in Figure 8.3, any plant with a feature vector that falls above the line will be classified as type B. Similarly, any plant with a feature vector that falls below the line will be classified as type A.

With this classification rule, we can correctly classify almost all of our training instances. However, note that we're not classifying perfectly. There is one rogue type B close to the rest of the type A's. In general, though, we will need to accept more error than this.

feature is sufficient to separate the two clusters. One can see this by *projecting* the scatter plot on to either one of the axes. When we do so, we see that the two classes are intermingled, rather than forming distinct clusters.

[3]There are a wide variety of possible distance metrics; Euclidean distance is certainly not the only choice.

[4]The two calculations actually differ by a scaling factor, since RMS involves a *mean* while Euclidean distance involves a *sum*.

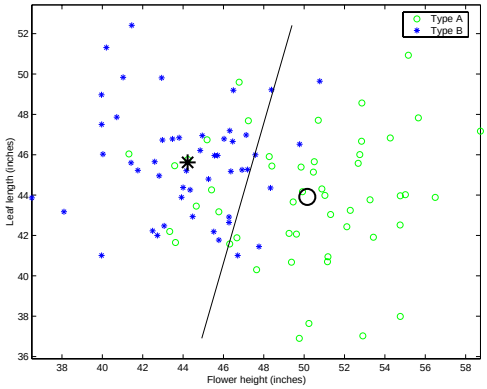

Figure 8.4: An example where two features are not as clearly separated.

Of course, two features may not be enough either. If our scatter plot looked like the one in Figure 8.4, then we can still see the two clusters, but we can't perfectly distinguish them based only on these two features. The line we draw for our distance rule will properly classify most of the instances, but many are still classified incorrectly. Once again, we can either accept the errors that will be made or we can try to find another feature to help us better distinguish between the two classes. Unfortunately, visualizing feature spaces with more than two dimensions is rather difficult. However, the intuition we've built for two-dimensional feature spaces extends to higher dimensions. We can calculate mean feature vectors and distances in the roughly the same way regardless of the number of dimensions.

8.2.3 A few more classification examples

We've looked at a simple classification task with only two classes, but there are some more examples that are instructive. Consider Figure 8.5(A). In this example, the two clusters fall right on top of one another, so we will have very poor classification performance. This is an example where neither of the features assist classification performance very much. In this case, we need to find better features before we can have much luck with classification. Figure 8.5(B) shows a similar example. Here, feature 2 will help us to improve our classification performance but feature 1 will not. (Can you see why?) Note that it may be worse to have a second feature which is bad than to only have one (good) feature. Unfortunately, determining which features are good and which are bad is nontrivial when we have more than two (or three) features and can no longer visualize the data.

It is also important to realize that we may have more than just two classes in a classification problem. Figure 8.5(C) shows an example in which we have four classes that have distinct clusters in our feature space. The mean feature vectors are indicated on these plots with large markers. Again, we can use the same distance-based decision rule to classify instances. That is, we classify a given instance according to the class whose mean feature vector is closest to its feature vector. We have included approximate) decision lines on this plot which partition the feature space (i.e., the plane) into four pieces. These indicate which class a given feature vector will be classified as. Of course, multiple classes can be indistinct, too. Figure 8.5(D) shows an example for three indistinct classes. Here, the blue ($*$) and green (o) classes are reasonably distinguishable, but we cannot easily separate the red (+)

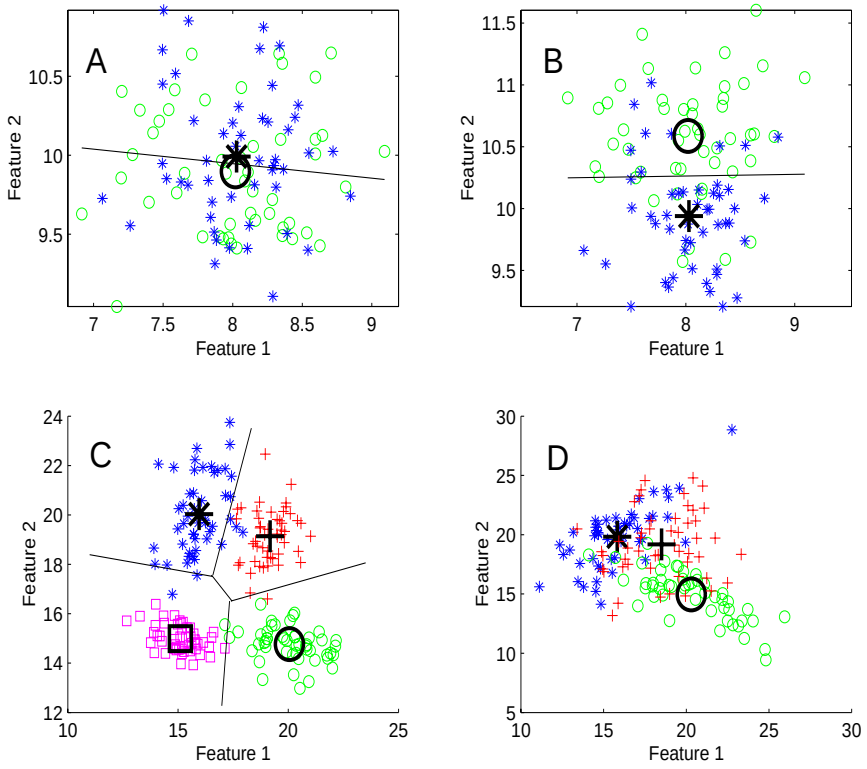

Figure 8.5: (A) Classes overlap, so the features do not allow much discrimination; these are bad features. (B) Feature 2 aids discrimination, but Feature 1 does not. (C) An example with four distinct classes; decision lines are approximate. (D) An example with three indistinct classes.

class from either of the other two.

8.2.4 Formalizing the feature classifier

In the previous sections, we presented some examples of classifier design and operation. Here, we'll formalize these ideas with respect to the general classifier block diagram shown in Figure 8.1. In particular, we will expand upon the *feature classifier* shown in that block diagram. Figure 8.6 shows an expanded block diagram of the feature classifier[5].

Suppose we need our classifier to decide among C classes, and suppose the classifier will be based on a set of N features, forming a feature vector $\mathbf{f} = (f_1, \ldots, f_N)$. Our feature classifier will rely on a set of representative feature vectors, one for each class, with the representative feature vector for the c^{th} class, denoted $\bar{\mathbf{f}}_c = (\bar{f}_{c,1}, \ldots, \bar{f}_{c,N})$, where $c = 1, \ldots, C$.

[5]The main goal of any feature classifier is to determine which of a set of representative feature vectors a new instance is most similar to. In this lab, we use *Euclidean distance* to measure similarity, and so we use a distance-based feature classifier. Other types of feature classifier are also possible, such as a correlation-based feature classifier.

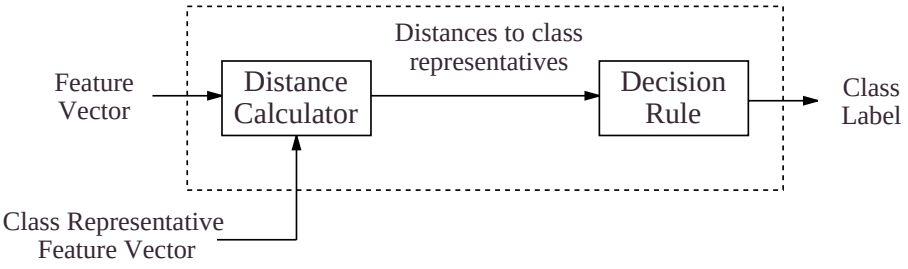

Figure 8.6: Block diagram of a distance-based feature classifier, which makes the decision in a general classifier system.

Given a set of representative feature vectors (the choice of such will be discussed later), we can classify new instances using the feature classifier. The feature classifier (seen in Figure 8.6) has two steps. The first step computes the distances between the input feature vector and each of the class representatives. As we have done in the previous sections, we will use Euclidean distance in our system. Equation (8.1) gives the formula for Euclidean distance in two dimensions. For a general, N-dimensional feature space, we use the following equation. Let $\mathbf{u} = (u_1, \ldots, u_N)$ and $\mathbf{v} = (v_1, \ldots, v_N)$ be two N-dimensional vectors (i.e., arrays with length N). We calculate the Euclidean distance between them as

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{N} (v_i - u_i)^2} = \sqrt{(v_1 - u_1)^2 + (v_2 - u_2)^2 + \cdots + (v_N - u_N)^2}. \tag{8.2}$$

Again, we note that, to within a scaling factor, Euclidean distance is equivalent to the root mean squared error between two vectors.

The second step of the feature classifier applies a *decision rule* to select the best class for the input instance. The decision rule that we will use is the *nearest class representative rule*. This simply means that the classifier *decides* the class whose whose representative feature vector is closest (in Euclidean distance) to the feature vector of the instance being classified. That is, if $\mathbf{f}$ is the feature vector for an instance to be classified, then the decision rule decides class c if $d(\mathbf{f}, \overline{\mathbf{f}}_c)$ is less than $d(\mathbf{f}, \overline{\mathbf{f}}_{c'})$ for all other classes c'[6]. Other decision rules, which may weight the distances from the various class representatives, are also possible, but they will not be considered here.

Note that our DTMF signal classifier from Lab #7 used a simpler "feature classifier" that was based on neither distance nor correlation. However, with a little extra work it could have been formulated as either of these types of classifier, most likely without a degradation of performance.

Let us now discuss how to choose the class representative feature vectors $\overline{\mathbf{f}}_1, \ldots, \overline{\mathbf{f}}_C$. Finding these is the main aspect in feature classifier *design*. We have previously suggested that we can find a representative feature vector for a class by taking the mean across some set of instances that belong to that class. We describe this calculation formally as follows. Suppose that we have a set of N-dimensional feature vectors from M instances of a given class c (this the design set of instances for this class). Let $\tilde{\mathbf{f}}_i = (\tilde{f}_{i,1}, \tilde{f}_{i,2}, \ldots, \tilde{f}_{i,N})$ denote

[6]If it should happen that $\mathbf{f}$ is equally closest to two or more class representatives, then an arbitrary choice is made among them.

the i^{th} such feature vector. We calculate the mean feature vector, $\bar{\mathbf{f}}_c = (\bar{f}_{c,1}, \ldots, \bar{f}_{c,N})$, for this class as

$$\bar{\mathbf{f}}_c = \frac{1}{M}\sum_{i=1}^{M} \tilde{\mathbf{f}}_\mathbf{i} = \frac{1}{M}(\tilde{\mathbf{f}}_\mathbf{1} + \tilde{\mathbf{f}}_\mathbf{2} + \cdots + \tilde{\mathbf{f}}_\mathbf{M}). \tag{8.3}$$

Alternatively, we can say that the j^{th} element of the mean feature vector, $\bar{\mathbf{f}}_c$, is

$$\bar{f}_{c,j} = \frac{1}{M}\sum_{i=1}^{M} \tilde{f}_{i,j} = \frac{1}{M}(\tilde{f}_{1,j} + \tilde{f}_{2,j} + \cdots + \tilde{f}_{M,j}). \tag{8.4}$$

8.2.5 Measuring the performance of a classifier

The performance of a classifier is based on the how many errors it makes. One good way to characterize the performance of a 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,

$$K_{i,j} = \frac{\#\text{ of class } j \text{ instances classified as } i}{\#\text{ of class } j \text{ instances}} \tag{8.5}$$

For example, the following is confusion matrix for a hypothetical four-class classifier:

$$K = \begin{bmatrix} .9 & .03 & .01 & .02 \\ .03 & .95 & .01 & .03 \\ .05 & .01 & .96 & .1 \\ .02 & .01 & .02 & .85 \end{bmatrix} \tag{8.6}$$

The diagonal elements, $K_{n,n}$, show what fraction of instances from the the n^{th} class were correctly classified. The .9 in the upper left corner, for instance, indicated that 90% of instances from the first class were classified as belonging to the first class. Thus, higher diagonal elements are desirable.

The off-diagonal element $K_{n,m}$ indicates what fraction of instances from the m^{th} class were misclassified as belonging to class n. In the example above, for instance, the .02 in the upper right corner indicates that 2% of instances in the fifth class were incorrectly classified as belonging to the first class. Thus, we hope that off-diagonal elements are as small as possible. The confusion matrix for a perfect classifier will be an identity matrix (i.e., ones on the diagonals, zeros elsewhere).

Data usage when designing 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*, than the one on which it was designed. 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.

To keep things simple, in this lab we will not separate our set of instances into separate design sets and testing sets. Thus, it is important to know that we may not be accurately characterizing our system's performance in the "real world." You will be given an opportunity to test our vowel classifier and see how well it actually performs on your voice. Specifically, all of the vowel instances provided in this lab were taken from a single speaker. How does this affect the performance of the system for *other* speakers? Can you come up with a better set of representative feature vectors (possibly by collecting vowel samples from a variety of speakers)?

8.2.6 Vowel Classification

So far, we have discussed a general-purpose framework for performing classification. In this section, we will specifically discuss how to apply these techniques to the classification of vowels in speech signals.

Vowels in speech are nearly periodic segments of the speech signal. From our studies of the Fourier Series, we know that these segments of the signal are thus approximately equal to the sum of sinusoids with harmonically related frequencies. As with the DTMF signals in Lab #7, the time-domain provides relatively little information about the signal. So, as with the DTMF signals, this suggests that we need to examine vowels in the frequency domain. Figure 8.7 shows examples of the magnitude spectrum (in decibels[7]) of two different vowels. The plots on the left correspond to an "ee" vowel (as in the word *tree*), while the plots on the right correspond to an "ah" vowel (as in the word *father*). Also shown is a smoothed version of each spectrum, which shows its general trend.

There are a number of interesting things to note about these plots. First, we can see the peaks that correspond to the harmonics that make up the periodic signal. Notice that the peaks are spaced more closely in some plots than others, corresponding to a lower fundamental frequency and thus a longer fundamental period. As illustrated by this figure, though, the fundamental frequency of the signal is independent of the vowel being produced. Notice that the overall shape of the frequency spectrum is different between the two vowels, but remains relatively constant between the two instances of each vowel, as can be seen from the smooth versions. This shape determines the *timbre*[8] of the sound, and, correspondingly, the "sound" of the vowel. Notice that there are peaks in the smoothed spectrum at various places. These peaks are called *formants*; it is generally known that the position of these formant is the primary feature that distinguishes one vowel from another, i.e. that makes one vowel sound different from another.

Unfortunately, there is no solid definition of a "formant," and they are remarkably difficult to identify automatically. In fact, there is some disagreement as to what constitutes a formant in some cases. In this lab, we'll work with two sets of features that hopefully capture the information contained in the formant positions. In Lab #9, we'll investigate

[7] To convert a number, x, into decibels, we use the formula $x_{dB} = 20 \log_{10}(x)$.
[8] Pronounced "tambor."

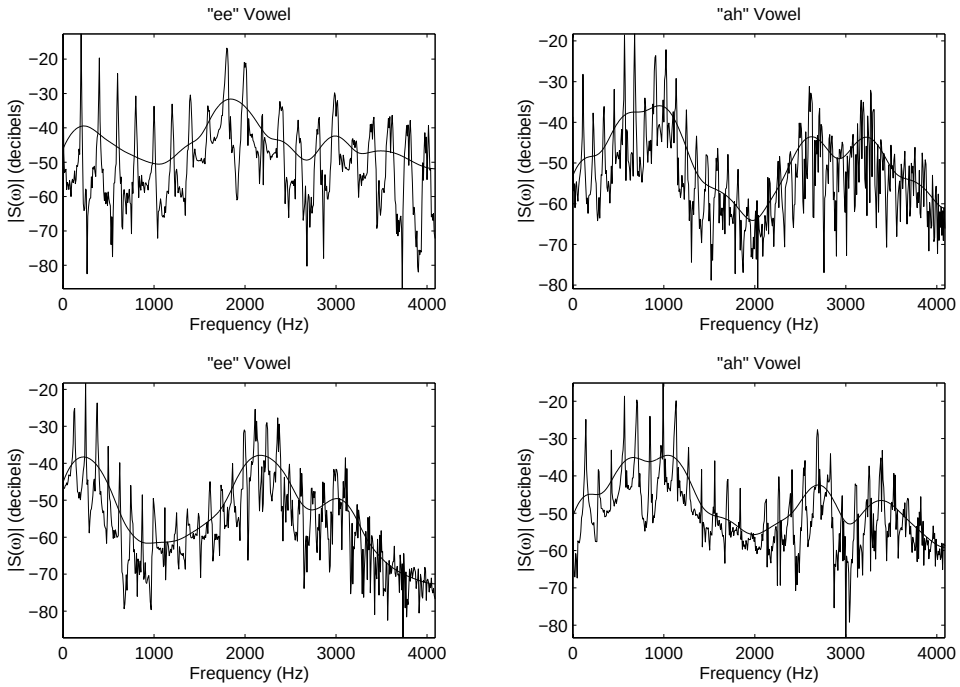

Figure 8.7: The magnitude spectrum (in decibels) of four vowel signals. The plots on the left correspond to two instances of an "ee" vowel, as in the word *tree*. The plots on the right correspond to two instances of an "ah" vowel, as in the word *father*. The solid line is a smoothed version of the spectrum, which shows the general trends of the spectrum.

the use of another, somewhat more sophisticated feature for vowel recognition. This feature actually models speech production, and thus should more readily capture the relevant aspects of the vowel signal.

The first feature set that we use in this lab will be the *formant features*. The formant features attempt to locate the formants themselves using a simple algorithm. This algorithm first uses the DFT to compute the spectrum of a short segment of a vowel. Then, the spectrum is smoothed using a weighted averaging filter. Finally, the algorithm returns frequencies of the largest peaks on the smoothed signal that occur above and below 1500 Hz. Thus, there are two formant features, so the resulting feature vector is two-dimensional.

The second feature set, the *filter bank features*, are quite similar to the features used in the DTMF decoder. The filter bank features compute the energy (in decibels) of the speech signal after it has been passed through a bank of six bandpass filters. We will use bandpass filters with center frequencies of 600 Hz, 1200 Hz, 1800 Hz, 2400 Hz, 3000 Hz, and 3600 Hz. Thus, the resulting feature vectors are six-dimensional.

Note that there are a large number of vowels that we could possibly consider. However, for simplicity we will restrict attention to just five vowels: "ee" (as in *tree*), "ah" (as in *father*), "ae" (as in *fate*), "oh" (as in *boat*), and "oo" (as in *moon*). Each of these five vowels will be its own class.

8.3 Some MATLAB commands for this lab

- **Converting a value into decibels:** Expressing a numerical value in *decibels* compresses the range of values using a logarithmic transformation. Thus allows us to see features that might otherwise not be visible. The decibel transformation is particularly useful when looking at the magnitude spectrum of audio signals, since hearing is based on a logarithmic amplitude scale. Given a value `x`, we convert it to decibels using the command

  ```
  >> x_dB = 20*log10(x);
  ```

  This command can be also used to simultaneously convert a vector of values to decibels.

- **Calculating features for a vowel signal:** As indicated, the features we would like to consider in order to classify a vowel signal are based on the signal's spectrum. We provide functions to calculate the two feature sets described in this laboratory. Each function takes an audio waveform, `x`, and (optionally) the sampling frequency in samples per second, `fs`. (If no sampling frequency is specified, a sampling frequency of 8192 samples per second is assumed.) Both functions return a row vector, `y`, that contains the features calculated from the waveform.

  To compute the "formant features," use `calc_formants.m`:

  ```
  >> y = calc_formants(x,fs);
  ```

  Similarly, to compute the "filter bank features," `calc_fbank.m`:

  ```
  >> y = calc_fbank(x,fs);
  ```

- **Working with features vectors in MATLAB:** In this lab, we will adopt the convention that a *feature vector* is a row vector, and that a set of feature vectors, such as a set of class representatives or a set of testing data, is stored in a matrix such that there is one feature vector per row and one feature per column. This allows us to easily compute mean feature vectors from such a matrix.

  When computing Euclidean distances, note that the computation is almost the same as that which we used for computing RMS error. The only difference is that we replace the *mean* operation by a *summation*.

- **Advanced plotting:** You may recall from Lab #1 that we can use MATLAB's `plot` command to change the color and style of plotted lines. A line-style string consists of as many as three parts. One part specifies a color (for instance, 'k' for black or 'r' for red). Another part specifies the type of markers at each data point (for instance, '`*`' uses asterisks while '`o`' specifies circles). The third part specifies the type of line used to connect the points ('`:`' specifies a dotted line, while '`-`' specifies a solid line). Note that these three parts can occur in any order, and all are optional. If no color is specified, one is chosen automatically. If no marker is specified, a marker will be not be plotted. If a marker is specified but a line type is not, then lines will not be drawn between data points. Thus, the command:

  ```
  >> plot(x1,y1,'rx',x2,y2,'k:');
  ```


will plot `x1` versus `y1` using red verb-x-'s with no connecting line, and also `x2` with `y2` with a dotted connecting line but no marker. See `help plot` for more details.

Additionally, we can change the width of lines and the size of markers using additional parameter-pairs. For instance, to increase the line width to 2 and the marker size to 18, use the command

```
>> plot(x1,y1,'rx--','Linewidth',2,'Markersize',18);
```

- **Executing the feature classifier:** The function `feature_classifer` is an incomplete function that you will use to automatically classify a feature vector (or a set of feature vectors) based on the distances to a set of representative feature vectors. The function takes two inputs. The first input, `M`, is a matrix of the representative feature vectors, with one feature vector per row. Note that the vector on the first row corresponds to the first class, the second row to the second class, and so on. The second input parameter, `fmatrix`, can either be a single feature vector to be classifed (stored as a *row vector*) or a matrix of feature vectors to be classified, with one feature vector per row. To call `feature_classifier`, use the command:

  ```
  >> labels = feature_classifer(M,fmatrix);
  ```

  The function outputs a *column vector* of class labels, `labels`, with one label for each row of `fmatrix`. The labels are numbers that indicate which representative feature vector the corresponding instance is closest to. Thus, if the first element of `labels` is a 3, it means that the feature vector in the first row of `fmatrix` is closest to the representative feature vector in the third row of `M`.

- **Calculating confusion matrices:** We provide you with a function, `confusion_matrix`, that computes a confusion matrix for you. Note that `confusion_matrix` calls your `feature_classifier` function, so it will not work until you have completed that function. To compute a confusion matrix, we need the matrix of representative feature vectors used by that classifier and a set of testing data for each class. Thus, if our matrix of representative feature vectors is `M` and `class1`, `class2`, and `class3` are matrices that contain feature vectors from our testing set with instances of each of three classes (with one feature vector per row), we compute the 3×3 confusion matrix using the command:

  ```
  >> K = confusion_matrix(M,class1,class2,class3);
  ```

  Note that the function `confusion_matrix` works for any number of classes. The size of the confusion matrix is determined by the number of input parameters.

8.4 Demonstrations in the Lab Section

- Introduction to Classification
- Evaluating a classifier
- Vowel spectra
- Vowel features

8.5 Laboratory Assignment

1. (Examining one vowel instance.) Download the file `lab8_data.mat`. This file contains a variable called `vowel1`, which is a one half-second recording of a vowel sound with a sampling frequency of 8192 samples per second.

   (a) Use `soundsc` to listen to this vowel.
   - [2] To which of the five vowel classes does this vowel belong?

   (b) Take the DFT of `vowel1`. In two subplots of the same figure, plot the magnitude of the DFT and the magnitude of the DFT in decibels. Only plot the *first half* of the DFT coefficients in each plot. Also, make sure that you label the x-axis with the frequency in Hertz, *not* the DFT coefficient number. (Hint: The maximum frequency showing on your plot should be 4096 Hz, which is one half of the sampling frequency.)
   - [4] Include this figure in your report.
   - [3] Compare the two plots. Are there aspects of the spectrum that are easier to see in one of the plots than in the other?
   - [2] From this figure, estimate the fundamental frequency of the vowel sound.
   - [2] Estimate the frequencies (in Hz) of the three most prominent formants.

   (c) Download the files `calc_formants.m` and `calc_fbank.m`. You will use them to calculate features for this vowel.
   - [2] Calculate and include the formant feature vector for this vowel.
   - [2] Compare the calculated features to your estimate of the formant locations.
   - [2] Calculate and include the filter bank feature vector for this vowel.
   - [2] What are the center frequencies of the filters that have the greatest output amplitude? Compare this to your estimated formant locations.

2. (Mean feature vectors and hand classification.) `lab8_data.mat` also has several other variables, including matrices containing features for 50 instances of each vowel. The variables `ah_form`, `ee_form`, `ae_form`, `oh_form`, and `oo_form` contain formant feature vectors for each vowel. Each matrix has 50 rows (one for each instance) and two columns (one for each feature value). Similarly, the variables `ah_fbank`, `ee_fbank`, `ae_fbank`, `oh_fbank`, and `oo_fbank` contain the filter bank feature vectors with one row per instance.

   (a) Calculate the mean feature vectors for each vowel class and for both feature classes.
   - [4] Include the five mean formant feature vectors in your report. Make sure you label them.
   - [4] Include the five mean filter bank feature vectors in your report. Again, make sure you label them.

   (b) (Generate a scatter plot) We would like to see how separable the classes are from the formant feature vectors. To do this, you'll create a scatter plot that plots the first formant location versus the second formant location. Plot each of the feature vectors, using a different color and marker symbol for each vowel. Make sure you include a legend. Also, plot the mean vector for each class on your scatter plot.

(Hint: To make your mean vectors stand out, you should increase the line width and marker size for just those points.)

- [10] Include the scatter plot in your report.
- [4] Interpret this scatter plot. Are all of the classes distinct and easily separated? Do you expect expect any vowels to be frequently confused? Do you expect any vowels to frequently be classified correctly?

Food for thought: What would happen if we only used one of these two features for classification? Which would give us better classification results? Do you think that a third formant feature might improve class separation?

(c) (Hand classification) Now, you'll "classify" the signal `vowel1` by hand. To do this, you'll need to compute the distance between the instance and the five mean feature vectors.

- [5] Compute the distances between the formant feature vector that you generated for `vowel1` and the five mean formant feature vectors.
- [5] Compute the distances between the filter bank feature vector that you generated for `vowel1` and the five mean filter bank feature vectors.
- [3] Using the nearest-representative decision rule, use the above results to classify `vowel1`. Do the results for both feature sets agree? If not, which feature set produces the correct answer?

3. (Complete and use the feature classifier code.) In this problem, you'll complete and then use a function, called `feature_classifier.m`, that does this classification for us automatically. As described in the background section, the function takes as input a matrix of representative feature vectors and a matrix of instances to classify. We output a label for each of the instances in the input.

(a) Complete the function. (Hint: You should use two `for` loops. One loops over the rows of the matrix of test instances. Then, for each instance, loop over the rows of `M` and compute the distances. To make the classification for each instance, find the *position* of the smallest distance and store it in `labels`.)

- [12] Include your code in your report.

(b) (Test `feature_classifier` on the formant features.) Place your mean formant feature vectors into a matrix, `M_form` with one feature vector per row. For consistency, put "oo" in the first row, "oh" in the second row, "ah" in the third row, "ae" in the fourth row, and "ee" in the fifth row. Call your `feature_classifier` function using this matrix and `ee_form`. (Hint: To make sure your function works correctly, you should compare its output to the completed and compiled function `feature_classifier_demo.dll`. If you did not successfully complete `feature_classifier`, you can use this demo function throughout the remainder of the lab.)

- [2] What fraction these instances are properly classified?
- [3] Calculate the fraction of the instances that are misclassified as each of the incorrect classes. That is, determine the fraction that are misclassified as "ah," the fraction misclassified as "ae," and so on.
- [1] From this data, what vowel is "ee" most often misclassified as?

(c) Repeat the above with the filter bank features, this time using the matrix `ee_fbank` and generating the matrix `M_fbank`. Use the same order for your classes. (Again, you should compare your function's output to the output of `feature_classifier_demo.dll`).

- [2] What fraction these instances are properly classified?
- [3] Calculate the fraction of the instances that are misclassified as each of the incorrect classes.
- [1] From this data, what vowel is "ee" most often misclassified as?

4. (Compute and interpret confusion matrices.) Download the file `confusion_matrix.m`. In this problem, you will compute and interpret confusion matrices for the two feature classes.

(a) Use `confusion_matrix` to compute the confusion matrix for the formant features. Use `M_form` as your set of class representatives. Use the vowel following vowel order for your remaining input parameters: "oo," "oh," "ah," "ae," and "ee." (This should be the same as the order as the classes in `M_form`).

- [3] Include this confusion matrix in your report. Label each row and column with the corresponding class.
- [2] In Problem 3b, you computed a portion of the confusion matrix. Identify that portion and verify that your results were correct.
- [2] From this confusion matrix, determine how many instances of "ee" vowels were misclassified as "oo" vowels.
- [1] Which vowel is most commonly misclassified using this feature set?

(b) Use `confusion_matrix` to compute the confusion matrix for the filter bank features. Use `M_fbank` as your set of class representatives. Use the vowel following vowel order for your remaining input parameters: "oo," "oh," "ah," "ae," and "ee." (This should be the same as the order as the classes in `M_form`).

- [3] Include this confusion matrix in your report. Label each row and column with the corresponding class.
- [2] In problem 3c, you computed a portion of the confusion matrix. Identify that portion and verify that your results were correct.
- [2] From this confusion matrix, determine how many instances of "ae" vowels were misclassified as "ah" vowels.
- [1] Which vowel is most commonly misclassified using this feature set?

(c) Finally, compare the two confusion matrices.

- [2] Which feature set has the best performance overall?
- [2] Based on the performance of these classifiers, comment on the spectral similarities between the various vowels. That is, are any of the vowel classes particularly like any of the other vowel classes?

5. On the front page of your report, please provide an estimate of the average amount of time spent outside of lab by each member of the group.

Food for thought:

- As was mentioned in the background section, all of the vowel instances used here were taken from a single speaker. As such, this classifier will probably perform better on that speaker's vowels than on the rest of the population. The compiled functions `formant_classifier.dll` and `fbank_classifier.dll` let you test the classifier designed here on your own voice. When you execute either of these functions (which require no input parameters), MATLAB will immediately record one quarter-second from the microphone, compute the features, and classify the vowel. Use this function to test the classifier on your own voice for various vowels. Record how well it does on each vowel. Is performance better or worse than what is indicated by the confusion matrices you calculated?

- The above function also returns the set of features that it computed from the recorded vowel. If you collect these features into series of matrices of testing data, you can use `confusion_matrix.m` to formally compute the performance of this classifier on your voice.

- How well does the function work on *other* people's voices? Are there certain people for whom it works very well? Very poorly?

- The compiled functions listed above take an optional input parameter, which is a matrix of representative feature vectors. Since the current classifier is designed using only one speaker's vowels, maybe you can improve the performance by coming up with a better set of representative feature vectors. To do this, consider gathering a set of vowels from a number of different speakers and combining them into a set of mean feature vectors. Can you improve the performance of the system?

