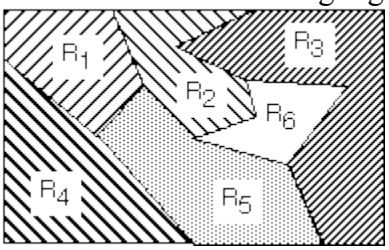
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2008
Recitation October 23

Learning to Learn: additional notes Bob Berwick

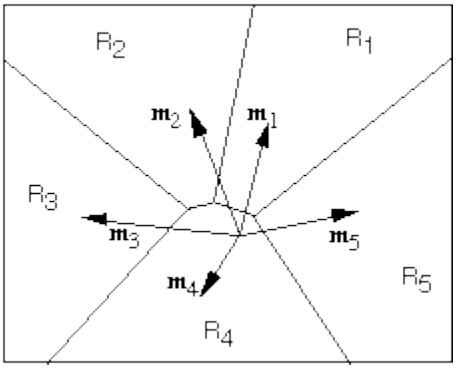
Agenda

1. **Learning: general issues**
2. **Features; Nearest Neighbors**
3. **ID trees: learning how to split**

1. In general, a pattern classifier carves up (or tesselates or partitions) the feature space into volumes called **decision regions**. All feature vectors in a decision region are assigned to the same category. The decision regions are often simply connected, but they can be multiply connected as well, consisting of two or more non-touching regions.

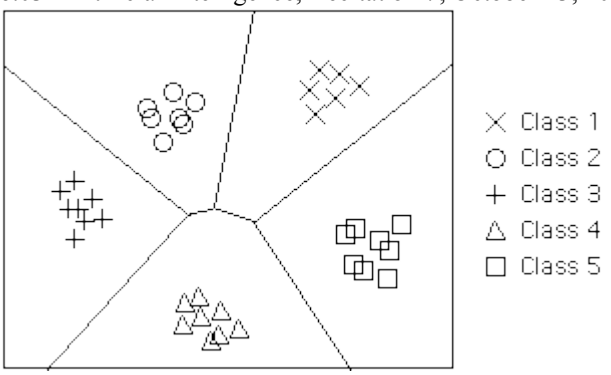


The decision regions are separated by surfaces called the **decision boundaries**. These separating surfaces represent points where there are **ties** between two or more categories. For a minimum-distance classifier like nearest neighbors, the decision boundaries are the points that are equally distant from two or more of the templates. With a Euclidean metric, the decision boundary between Region *i* and Region *j* is on the line or plane that is the perpendicular bisector of the line from $\mathbf{m}_i$ to $\mathbf{m}_j$. Analytically, these linear boundaries are a consequence of the fact that the discriminant functions are linear.



Nearest-template decision boundaries

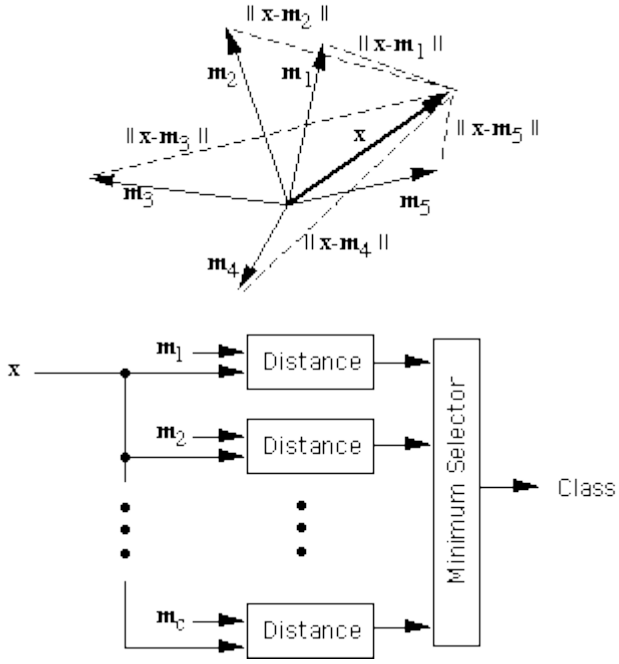
How well the classifier works depends upon how closely the input patterns to be classified resemble the templates. In the example sketched below, the correspondence is very close, and one can anticipate excellent performance. However, things are not always this good in practice, and one should understand the limitations of simple classifiers.



Template matching can easily be expressed mathematically. Let **x** be the feature vector for the unknown input, and let $\mathbf{m}_1$, $\mathbf{m}_2$, ..., $\mathbf{m}_c$ be templates (i.e., perfect, noise-free feature vectors) for the *c* classes. Then the error in matching **x** against $\mathbf{m}_k$ is given by

$$\| \mathbf{x} - \mathbf{m}_k \| .$$

Here $\| \mathbf{u} \|$ is called the **norm** of the vector **u**. A minimum-error classifier computes $\| \mathbf{x} - \mathbf{m}_k \|$ for $k = 1$ to *c* and chooses the class for which this error is minimum. Since $\| \mathbf{x} - \mathbf{m}_k \|$ is also the distance from **x** to $\mathbf{m}_k$, we call this a **minimum-distance** classifier. Clearly, a template matching system is a minimum-distance classifier.



There is more than one way to define the norm $\| \mathbf{u} \|$, and these correspond to different ways of measuring distance, i.e., to different **metrics**. Two of the most common are

1. **Euclidean** metric: $\| \mathbf{u} \| = \text{sqrt}(u_1^2 + u_2^2 + ... + u_d^2)$
2. **Manhattan** (or taxicab) metric: $\| \mathbf{u} \| = |u_1| + |u_2| + ... + |u_d|$

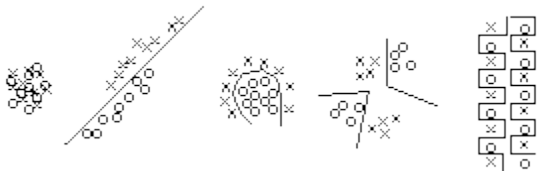
In our template-matching example of classifying characters by counting the number of disagreements, we were implicitly using a Manhattan metric. For the rest of these notes, we will use either the Euclidean distance or something called the Mahalanobis distance. We will see that

- Contours of constant Euclidean distance are circles (or spheres)
- Contours of constant Manhattan distance are squares (or boxes)
- Contours of constant Mahalanobis distance are ellipses (or ellipsoids)

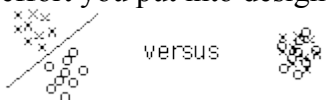
Limitations

If a simple minimum-distance classifier is satisfactory, there is no reason to use anything more complicated. However, it frequently happens that such a classifier makes too many errors. There are several possible reasons for this:

1. The features may be inadequate to distinguish the different classes
2. The features may be highly correlated
3. The decision boundary may have to be curved
4. There may be distinct subclasses in the data
5. The feature space may simply be too complex

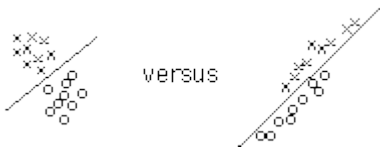


1. **Inadequate features:** If the features simply do not contain the information needed to separate the classes, it doesn't matter how much effort you put into designing the classifier.



Solution: go back and design better features.

2. Correlated features. It often happens that two features that were meant to measure different characteristics are influenced by some common mechanism and tend to vary together. For example, the perimeter and the maximum width of a figure will both vary with scale; larger figures will have both larger perimeters and larger maximum widths.



This degrades the perfomance of a classifier based on Euclidean distance to a template. A pattern at the extreme of one class can be closer to the template for another class than to its own template. A similar problem occurs if features are badly scaled, for example, by measuring one feature in microns and another in kilometers.

Solution: Use the Mahalanobis metric (what's that??)

The use of the Mahalanobis metric removes several of the limitations of the Euclidean metric:

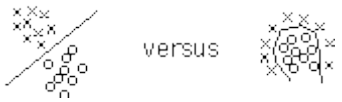
1. It automatically accounts for the scaling of the coordinate axes
2. It corrects for correlation between the different features
3. It can provide curved as well as linear decision boundaries

You don't need to know about this for this course (but you should…). You can visualize this as a set of ellipses, as shown above. More precisely, The quantity r in

$$r^2 = (\mathbf{x} - \mathbf{m}_x)' C_x^{-1} (\mathbf{x} - \mathbf{m}_x)$$

is called the Mahalanobis distance from the feature vector **x** to the mean vector mx, where *Cx* is the covariance matrix for x. It can be shown that the surfaces on which *r* is constant are ellipsoids that are centered about the mean mx. In the special case where the features are uncorrelated and the variances in all directions are the same, these surfaces are spheres, and the Mahalanobis distance becomes equivalent to the Euclidean distance.
One can use the Mahalanobis distance in a minimum-distance classifier as follows. Let m_1, m_2, ... , m_c be the means (templates) for the *c* classes, and let C_1, C_2, ... , C_c be the corresponding covariance matrices. We classify a feature vector **x** by measuring the Mahalanobis distance from **x** to each of the means, and assigning **x** to the class for which the Mahalanobis distance is minimum.

4. Curved boundaries. The linear boundaries produced by a minimum-Euclidean-distance classifier may not be flexible enough. For example, if x_1 is the perimeter and x_2 is the area of a figure, x_1 will grow linearly with scale, while x_2 will grow quadratically. This will "warp" the feature space and prevent a linear discriminant function from performing well.

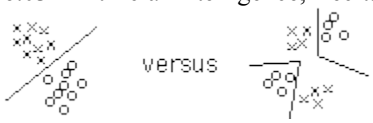


Solutions:

1. Redesign the feature set (e.g., let x_2 be the square root of the area)
2. Try using Mahalanobis distance, which can produce quadratic decision boundaries (see later, or maybe, not here – actually, see the end of these notes)
3. Try using a neural network or a support vector machine (see later in the course)

4. Subclasses. It frequently happens that the classes defined by the end user are not the "natural" classes. For example, while the goal in classifying English letters might be just to decide on one of 26 categories, if both upper-case and lower-case letters will be encountered, it might make more sense to treat them separately and to build a 52-category classifier, OR-ing the outputs at the end. Of course, it is not always so obvious that there are distinct subclasses.

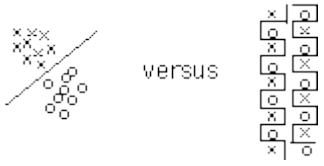
Solution: Use a **clustering procedure** to find subclasses in the data, and treat each subclass as a distinct class.



4. Complex feature space. It can happen that the variability that makes it difficult to distinguish patterns is due to **complexity** rather than noise. For example, it would make no sense to try to use a nearest-distance classifier to detect syntactic errors in expressions in a programming language.

Similar problems arise in classifying sensory data. Whenever the patterns can be subject to well-defined transformations such as translation or rotation, there is a danger of introducing a large amount of complexity into a primitive feature space. In general, one should employ features that are **invariant** to such transformations, rather than forcing the classifier to handle them.

In human communication, larger patterns are frequently composed of smaller paterns, which in turn are composed of smaller patterns. For example, sentences are made up of words which are made up of letters which are made up of strokes.



Validation.
One of the key questions about any classifier is how well it will perform, i.e., what is the **error rate**.

One way to estimate the error rate is to see how well the classifier classifies the examples used to estimate the parameters. This is called **testing on the training data**. It is a good necessary condition – if the classifier cannot correctly classify the examples used to train it, it is definitely worthless. However, it is a very poor measure of generalization ability. It inevitably promises much better performance than will be obtained with independent test data.

One simple validation procedure is to divide the available data into two disjoint subsets -- a subset used to train the classifier, and a subset used to test it. This is called the **holdout method**. It works reasonably well, but it often results in suboptimal performance because if you holdout enough examples to get a good test you will not have enough examples left for training.

There are several other more sophisticated alternatives. For example, in ***k*-fold cross-validation**, the examples are divided into k subsets of equal size. (A common choice is $k = 10$.) The system is designed k times, each time leaving out one of the subsets from training, and using the omitted subset for testing. Although this approach is time consuming, most of the data can be used for training while still having enough independent tests to estimate the error rate.

In a related approach called **bootstrapping**, one forms the the test set by randomly sampling the set of examples.