CSE 6390D 2010 (F)
Prof. J. Elder
Assignment 2
Due Date: Wed Dec 1, 2010 (Beginning of Class)

Overview

In this assignment, we will apply our understanding of machine learning and pattern recognition to the problem of shape classification. We will restrict our attention to planar (two-dimensional) shape, in particular to the boundaries of animal models.

Dataset

The dataset is drawn from the Hemera database of 150,000 blue-screened photo-objects. From these I have selected 350 animal objects and 350 vegetable objects, randomly partitioning them into training and test datasets of 175 objects each. The boundary of each object has been down-sampled to a vector of $D = 128$ points. Each point of a shape is a 2D Euclidean coordinate. We represent this as a complex number $x + iy$. The data and code I provide uses this representation, as it makes the code a little simpler.

Each shape has been normalized to a unit circle using a Procrustes transformation. For the purposes of this assignment it is not necessary to understand this normalization process, but there are two crucial implications:

1. There is a 1:1 correspondence between the 128-element vectors representing each shape, which facilitates analysis.
2. The expected position of a point on a shape is given by the corresponding point on the unit circle: $E\left[\left(x_i, y_i\right)\right] = \left(\cos\theta_i, \sin\theta_i\right)$, where $\theta_i = \frac{2\pi i}{D}$.

You can access the training dataset now from the course website.

Code
I have provided code for 2 classifiers:

Model 1. K-Nearest-Neighbours. This selects the class based upon a majority vote of the k neighbours nearest the test shape, measured by the Euclidean distance.
Function:

- ShapeClassifier1

Model 2. Least-Squares. This model uses the 1-of-K binary coding scheme and fits a hyperplane to the training data, selecting the class with maximum predicted value (see Ch. 4.1.3).
Function:

- ShapeClassifier2

In addition, I have provided the following utility functions:

- **ShapeClassSplit.m**. Given shapes from the two classes, and a parameter ptest that determines the proportion of data reserved for validation, this function outputs randomly-ordered training and test data, with classes (0 or 1) explicitly represented. The models above expect data in this format.
- **EvaluateShapeClassifiers.m**. Calls several classifiers and plots their performance.

This code can now be downloaded from the website.

Tasks

1. First start by downloading the dataset and code. Make sure that you can run the code and that you are not missing any required toolbox functions. First run ShapeClassSplit to partition the training dataset. Then try running the Models 1 and 2. Finally, try running the evaluation function to plot performance.
2. Now go through the code in detail and make sure you understand each step. It is often helpful to use the debugger to step through line-by-line, examining data structures along the way.
3. Now for the fun part: your goal is to create a new algorithm that performs better than either of the models I have given you. You can create as many as you like, and document them in your report, but only one will be evaluated on the test set.

Hints

1. In my brief investigations, I have found it challenging to do much better than the nearest-neighbour algorithm. However, there are many things I haven't yet tried. Keep in mind that you do not have to beat it to get a good mark, but you do have to correctly implement reasonable alternative algorithms.
2. Start by trying sensible things straight out of the textbook. For example:
    a. Fisher's linear discriminant
    b. Probabilistic generative model
    c. Logistic regression
    d. Kernel methods
    e. Sparse kernel methods
3. Remember that overlearning can be as much a problem here as it was for shape completion. Thus reducing the dimensionality of the input can improve performance!

Grading: This assignment is worth 20% of the course mark. Grades will be determined by the quality of the report submitted and the accuracy of your algorithms. You also get points if you find any bugs in code I provide!!

On the Complex Representation: In our representation of a planar shape as a D-dimensional complex vector, the complex covariance only has $2D(D+1)/2 = D(D+1)$ degrees of freedom. This is insufficient to represent the general covariance between the 2*D* scalars that define each shape. Thus the complex representation entails an assumption that the vectors obey certain symmetries. The specific assumption we are making is that the shapes are *proper* complex random vectors.

Proper Complex Random Vectors

Let $\mathbf{z} = \mathbf{x} + i\mathbf{y}$ be a $D \times 1$ proper complex random vector. Then $\mathbf{z}$ satisfies:

$$E\left[\left(x_i - E\left[x_i\right]\right)\left(x_j - E\left[x_j\right]\right)\right] = E\left[\left(y_i - E\left[y_i\right]\right)\left(y_j - E\left[y_j\right]\right)\right]$$

and

$$E\left[\left(x_i - E\left[x_i\right]\right)\left(y_j - E\left[y_j\right]\right)\right] = -E\left[\left(y_i - E\left[y_i\right]\right)\left(x_j - E\left[x_j\right]\right)\right]$$

For our shape representation, this means that for any two points, we assume that the covariance between their horizontal positions is the same as the covariance between their vertical positions, and that the covariance between the horizontal position of the first point and the vertical position of the second point is the negative of the covariance between the vertical position of the first point and the horizontal position of the second point.

These are quite strong assumptions. So one way to try to improve prediction is to change to a real representation, where a shape is now represented as a 2D-dimensional real vector. Then the covariance matrix has $2D(2D+1)/2 = D(2D+1)$ degrees of freedom, and these symmetries need not be assumed.

Of course, you may be struck by the curse of dimensionality! But we have ways of dealing with that...