Clustering and Classification

Chris Brown

April 7, 2005

1 Explanation

This is actually an assignment from the computer vision class. It sort of shows what a possible classification project would be like, and it has a tutorial about clustering techniques. Thus I thought it might serve as useful extra reading in 242. [CB April 7 2005].

2 The Problem

Your problem is to classify input images into some small number (say 5-10) of classes. You get to choose your inputs and your features. As always, you get to choose your programming environment: for the clustering and classification you only need arrays, but there's the image processing as well. The general scenario is that you will train your recognizer with examples and after training it will classify some previously unseen inputs and you will report on its performance.

Sample domains and classes might be

INPUT	CLASSES	FEATURES
color patches	red, yellow, blue, ..., brown (not easy)	(hsv, rgb...)
texture image patches	grass, sand, concrete, bricks, hair, plaid,...	Laws texture energy, edge density and direction...
faces	old, young or male, female or happy, sad, scared, angry, bored...	pixel values of whole input (!), region shapes & relations
digits	0,1,...9	shape measures, holes, st. lines, terminations, etc.
kids' blocks	arch, bar, brick, half-round, star	shape measures

toys	plane, doll, snake, car,...	shape measures,
photos	inside, outside or rightside up, upside down or party, christmas, vacation, sunset, portrait	color, texture, relations of color patches, shapes, faces...

OK, you get the idea.

As usual, we want a written report explaining your method and the assumptions underlying it, results in the form of statistics or graphs (confusion matrices are good), suggestions for future enhancements, final conclusions, and references if any. Your data can be made up by you (text editor, lab camera) or there are datasets (digits, faces, scenes) on the web. Further, `/u/brown/DigitData/optdigits-orig.tra` is a database of binarized cleaned normalized unthinned handwritten digits.

There is a lot written on pattern recognition, and if you want to cite some of the literature to compare your algorithm to other existing ideas, fine. However, this is not meant to be a library research exercise as much as a thinking exercise.

3 Classification

You will implement clustering, with at least three classification methods: nearest-centroid, nearest neighbor, and k-nearest neighbor. Scientifically compare their performances. If you want to write a decision tree and compare that, fine. If you want to import or write neural net classification code, fine: but be aware you may need many (many) training samples.

Clustering isn't mysterious. If you have N features, each with a real-number output (e.g. density of 1's, or number of diagonal edges in the upper left quarter of the images, or the B values of a B-binned edge-orientation histogram, or Fourier components of boundary shapes, or (R,G B) (or (H,S,V) or other color space coordinatees), or WHATEVER N features, they're just an N-vector. You may want to normalize the values so that the amount they contribute to the distance metric will be scaled right. Scaling by their individual standard deviations is a good start.

Anyway, during training you can just keep the average of these vectors for each class and that will be the mean vector (centroid) for that cluster. Then to test, you get a new vector and calculate its Euclidean distance from these class-representative mean vectors and the nearest one is your classification.

Alternately, remember all the training vectors (in an array say) along with their classification (in a parallel array). For a new test vector, calculate which of all the training vectors is closest, and its class is your answer. This is "nearest-neighbor". K-nearest neighbor is simple too, and you may want to try other tricks like principal component analysis (PCA) to throw out useless features, etc. etc. PCA isn't that hard, you just need to multiply your data matrix by itself and give it to a one-line MatLab command. See me if you need more info.

If you think about decision trees you'll see that they ask a sequence of questions, each of which divides feature space into two pieces. So they are another way of getting a decision surface in feature space. My guess is that it takes a fair amount of ingenuity to devise a tree that can cope with oddly-shaped or closely-intertwined clusters.

There are *lots* of clustering algorithms: you could look up and implement "k-means", for instance, which is a fun iterative one.

4 Features

It's the features, stupid! In other words the most sophisticated classification algorithm is going to have trouble if the features don't separate the classes, and if they do then the simplest technique should work very well.

In any event your choice of features should reflect the characteristics you are trying to separate. Ideally you can do well by just thinking, but sometimes one just heaves all possible features into the mix and then prunes out those that don't perform (using PCA). The features are the computer vision part of the work, and can be found in your text in the relevant chapter.

For some domains you may want to consider features that offer invariance to systematic geometric variations, like: Translation, Rotation, Size variation (zoom), Skew (say italics vs. roman or different handwriting slants).

You may want to use preprocessing to deal with noise: high-frequency (speckle in the background), low-frequency (blotchy low-contrast background variation), dropout (holes where there should be letter).