COMP4702/COMP7703 - Machine Learning

Prac 3 – Multivariate Parametric and Nonparametric Density Estimation

Aims:

- To complement lecture material in understanding the operation of nonparametric techniques.
- To gain experience with simulating and implementing these techniques in software.
  → To produce some assessable work for this subject.

Procedure:

Multivariate Parametric Techniques

On the course website you will find a copy of the Pima Diabetes dataset – a commonly used dataset for evaluating classification techniques. This dataset describes a two-class classification problem, with 8 input features and 768 data points.

→ **Q1:** Apply and evaluate quadratic discriminant analysis on the diabetes dataset.

→ **Q2:** Apply and evaluate linear discriminant analysis on the diabetes dataset. If the last digit of your student number is:
(i) 0-3: Use a full but common/shared covariance matrix for each class.
(ii) 4-6: Use a diagonal common/shared covariance matrix for each class.
(iii) 7-9: Use a diagonal common/shared covariance matrix with equal variances for all dimensions, for each class.

Details/hints:
→ Split the data into training (the first 500 points) and test (the remaining points) sets.
→ Fit the model using the training set.
→ Report (a) the training classification error; (b) the test classification error; (c) the model parameters (mean vectors and covariance matrices). To calculate the classification error, assume a decision threshold of 0.5 on the posterior probability values and then calculate the "percentage correct" for the data set in question. Note this corresponds to a 0/1 loss function after the decision threshold converts the prediction to a 0/1 output.
→ For Q2, you will need to pool the data to estimate the covariance matrix (see Alpaydin, Sec.5.5 for how to do this).

Nonparametric Density Estimation

Some of the techniques discussed in lectures are very widely used. In this part of the prac we will experiment with:

- **Histograms:** in Matlab as function hist(). Type "doc hist" for the appropriate entry in the very useful Matlab help documentation.

- **Kernel density estimation:** also in Matlab – in the Help browser see "Statistics Toolbox -> Descriptive statistics -> Percentiles and Graphical Descriptions -> Probability Density Estimation", and also "doc ksdensity".

→ Firstly we need some data. Import the fourth attribute (petalwidth) from the Iris dataset used in an earlier prac into a matlab variable. What follows assumes this data is stored in a vector called 'iris'.

→ **Q3:** Read the matlab documentation for the hist function. Produce a histogram of this data (with the default 10 bins) – hist(iris). Experiment with different numbers of bins (e.g. 2,4,8,10,20,50,100,...), observe the result and make sure you understand what is happening. What happens when the number of bins approaches (and becomes greater than) the number of data points?

→ Read the matlab documentation for the ksdensity function. Produce a kernel estimator of the iris data:
  - [f,x,h]=ksdensity(iris);
  - plot(x,f)

→ What is happening here is that matlab is using a Gaussian kernel, with width 'h' parameter chosen automatically (don't worry too much about how). What is returned are the density values (f) at 100 equally spaced points (x) in the range of the iris data. Also, the value of u gives the value of the width parameter.

→ **Q4:** Explore the effect of the width parameter in the estimator by producing a plot to compare (on the same axes): a number of estimators visually:
  - The default estimator (note the h value)
  - One estimator with larger than the default width
  - Two estimators with smaller than the default width.

It is often not easy to evaluate the "goodness" of a density estimator quantitatively, since we do not know the true distribution which produced the data. The situation is worse than for supervised learning because we don't even have a labelled set of data (which in this case would be a set of data points together with the true value of the density at each point). One approach (as mentioned in lectures and the book) is to measure likelihood on a validation set.

On the other hand, if we're just testing algorithms then we can generate our data from a *known* distribution. If we then build a density estimator from such data, we can compare the estimator with the true distribution that generated the data. A commonly used measure to compare probability distributions is the Kullback-Leibler (KL) divergence:

$$D_{KL}(p \parallel q) = \sum_i p(x_i) \log \frac{p(x_i)}{q(x_i)}$$

where p(x) is the true distribution and q(x) is the model distribution (see e.g. wikipedia for more info).

→ Generate a dataset from a mixture of two Gaussians:
  - x = [randn(30,1); 5+randn(30,1)];

→ Build a histogram model (H1) for this data with 20 bins.

→ Build two kernel density estimators (K1 and K2) for the data: one with the default kernel width value and one with a width value equal to half the default value.

→ **Q5:** Calculate the KL divergence (based on 100 equally spaced "test" points that cover the range of the data in x), between:
  - M and H1
  - M and K1
  - M and K2

  where M is the distribution given by the mixture of 2 Gaussians. In addition to the 3 KL divergence values, provide the code used to calculate them. Include and discuss any assumptions or "work-arounds" you used.

→ Some hints:
  - Recall mixture models from the clustering lecture: the distribution M above is a simple mixture with two components, each with a prior/mixing coefficient with value 0.5.
  - The matlab normpdf() function evaluates the density of a Gaussian at a given point.
  - The test points required are calculated via the ksdensity function.