Bioinformatics – Lecture Notes

Announcements

Reference: Microarray Data Analysis and Visualization by Arun Jagota

3 days of classes including today.

Final Projects due May 2 (Next Thursday). Please let me know your audio-visual need for your presentation

Class 28 – April 25, 2002

I. Classifying samples from two populations

A. Variable selection

We previously discussed methods that would identify genes that could be used to discriminate between the two tissues. The problem with these methods is that the treated each gene individually. We need methods that look at the genes collectively. The method we choose has to be able to 1) evaluate the quality of a subset of the variables and 2) search through all subsets of the data

One method is *sequential backward elimination*. This method starts from the full set of genes and removes genes one at a time on the basis of which gene make the smallest reduction (or largest increase) in the performance of the classifier among all candidates considered for removal

Another method is *greedy selection*. In this, we start with an empty set of genes and add the gene that makes the best one gene classifier. Then we add the gene that results in the best two-gene classifier, etc. This is faster if we are interested in obtaining a small number of classifiers.

B. Discriminant analysis

This method uses a *discriminant function* that defines a decision boundary.

Figure of *linear discriminant function*

A linear discriminant function has the form

$$f_{w,w_0}(x) = w^T x + w_0$$

where x is the point in R^n to be classified, and the vector **w** in R^n and the scalar w_0 parameterize the function. The point is classified a positive if $f(x)>0$ and negative otherwise.

A linear discriminant function can be thought of as a one layer neural network, the preceptron neural network, in which the outer layer only has one neuron. This neuron computes the step function

$$y(x) = \begin{cases} 1 & w^T x + w_0 > 0 \\ 0 & otherwise \end{cases}$$

The input layer has n neurons, one for each component of x. The neurons have the weights which are the components of **w**. The bias of the output neuron is given by w_0.

Training is done by a set that has elements that are known to lie on each side of the decision boundary. The data is presented sequentially. If the perceptron misclassifies a presented data element, the weights have to be adjusted, ie. y(x) does not equal the true label l(x),

$$w := w + (2l(x) - 1)x \qquad w_0 := w + (2l(x) - 1)$$

One would cycle through the data as many times as are need to classify the data correctly. After training the perceptron is given new data and should be able to classify it.

C. Multilayer perceptrons

The multilayer perceptron will have a hidden layer between the input and output layers. Again there is only one output neuron and the perceptron is trained on know data.

In the multilayer perceptron, information will travel from the input layer to the output layer in the following manner. Each hidden neuron will compute the weighted sum of all its inputs and then apply a *sigmoid transfer function* to convert this sum into a number between 0 and 1. After all the hidden neurons have done this, the output neuron will compute the weighted sum of all these values and then apply a sigmoid transfer function to convert this sum into a number between 0 and 1.

If this network had *h* hidden neurons then it computes a function $f_{w,W}(x)$ on an input x in R^n

$$f_{w,W}(x) = g\left(\sum_{j=1,\ldots,h} W_j g\left(\sum_{i=1,\ldots,n} x_i w_{ij} \right) \right)$$

$$g(u) = \frac{1}{1+e^{-u}}$$

where g(u) is the sigmoid transfer function, w_{ij} is the weight from input neuron i to hidden neuron j, and W_j is the weight from hidden neuron j to the output neuron.

D. Training

The neuron has to learn the weights w and W from the training set D that consists of data that is known to be in one class or the other. To do this, one can employ *gradient descents* to minimize an *error function*. One widely used error function is the *sum-of-squares*

$$E(w,W) = \frac{1}{2} \sum_{d \in D} \left(d.y - f_{w,W}(d.x) \right)^2$$

where d.y = 0 or 1 give the classification of training data point d and d.x is in R^n is its input vector. The weights will be varied to minimize E. This is done by choosing each weight $\boldsymbol{w}_i$, one at a time, and incrementing it by

$$\Delta \boldsymbol{w}_i = -\boldsymbol{h} \frac{\partial E}{\partial \boldsymbol{w}_i}$$

where $\boldsymbol{h} > 0$ is the *learning rate* and controls the magnitude of the increment.

Details available on page 73 of Jagota.

E. Validation

It is important to validate the network. To do this, before training break up the known data in D into a training set and a validation set. Train the network on the training set and test it on the validation set.

Split the data D into many different pairs of training and validation sets and pick the network that performs the best

Another method is called *k-fold validation*. To do this, break up the data into k equal sets. Train the data on the D minus the i^{th} set and test on the i^{th} set. Do this for i = 1,…,k. Average the performance over all these runs to give the performance of the network.

F. Overfitting

If you train a network too much, the error on the validation set will start to increase, even though the error on the training set keeps decreasing. This is because the network will be overly tuned to the training data. The way to prevent this is to stop training as soon as the error on the validation set starts to rise.

G. Support vector machines

This is a relatively new type of classifier and has a variety of applications. Define a classifier by the discriminant function

$$f(x) = w_0 + \sum_m w^m y^m (x^m \circ x)$$

where $\mu = 1,\ldots,m$ is a training set of examples, $\mathbf{x}$ is the example to be classified, $\mathbf{x}^\mu$ is in R^n is the input, and y^μ is either +1 or –1 and represents the binary label of $\mathbf{x}^\mu$. The dot product is used as a measure of similarity. The classification rule is

$$predictive\_label(x) = \begin{cases} +1 & if\ f(x) > 0 \\ -1 & if\ f(x) < 0 \end{cases}$$

If the coefficients w^μ are set to 1, then $f(\mathbf{x})$ is a weighted neighbors classifier with points $\mathbf{x}^\mu$ similar to x casting stronger votes for their labels y^μ than points dissimilar from $\mathbf{x}$. Note that this discriminant function defines a linear boundary.

Example – All negative values have $x \le 4$ and all positive values have $x \ge 6$. We need to determine a discriminant function that classifies the entire training set.

If the boundary that separates the data into two classifications is non-linear, we need a different discriminant function

$$f(x) = w_0 + \sum_m w^m y^m K(x^m, x)$$

where K is a measure of similarity called the *kernel* that computes a dot product in some space that is usually unknown. This means that K is a kernel function iff there exists a function f which maps R^n to some space H such that

$$K(x, y) = f(x) \circ f(y)$$

for all x and y in R^n. For example, the *polynominal kernel* is for some positive integer p $K(x, y) = (x \circ y)^p$. Another kernel is

$$K(x, y) = e^{-\|x-y\|^2/2s^2}$$

Training these requires a background in *convex programming* which is beyond the scope of this course. In summary, we have to determine the weights of the training examples so that the decision boundary is in the right place. To do this we want this boundary to be as far away from points near the boundary as possible. In effect, this sets up a margin around the boundary. The goal of training is to maximize this margin. However, in doing so, we would be learning the noise along with the data. To compensate for this, we can allow a soft margin in which we see a boundary that has the possibility of misclassifying some of the training set data.

Question: Does this remind you of something?

Example – page 83 from Jagota

Next time:

II. Identifying genes expressed differently in paired samples
III. Identifying genes expressed differently in more than two populations
IV. Gene regulation networks
    A. Combinatorial approach for parameter estimation
    B. Modeling dynamics for parameter estimation
    C. Regression for parameter estimation
    D. Bayesian networks for parameter estimation