Chapter 4

Related Works on Class Discovery and Class Prediction

4.1 Introduction

Class discovery refers to the process of dividing samples into reproducible classes that have similar behavior or properties, while class prediction places new samples into already known classes (Slonim *et al.*, 2000). Class discovery is an unsupervised learning method and clustering is the widely used method for it; class prediction is a supervised learning method and different machine learning classifiers like support vector machine (SVM)(Vapnik, 1995), C4.5 (decision tree) (Quinlan, 1993), k-nearest neighbor (kNN) (Dasarathy, 1991), naive-Bayes classifier (NBC), weighted voting classifier (WVC) (Golub *et al.*, 1999; Slonim *et al.*, 2000), artificial neural network (ANN), to name a few prominent classifiers, are used as class predictors. The steps for class prediction are as follows:

1. Divide the available samples into training and test subsets.
2. Build a class predictor (classifier) using the training data set. In actual implementation, the values of different parameters are learnt using the training data in this step.
3. Using the test data set, evaluate the performance of the classifier built in previous step.

Sometimes, to extract an informative gene subset using a wrapper approach, the training data set is divided into internal training and validation subsets. For a

gene subset, the internal training subset is used to build the classifier and the internal validation subset is used to calculate the fitness.

4.2 Clustering of samples for class discovery

A widely used technique for microarray data analysis is clustering of the samples (Eisen *et al.*, 1998; Ben-Dor *et al.*, 1999; Alon *et al.*, 1999; Alizadeh *et al.*, 2000). In most cases, clustering is applied on reduced dimensional data after selecting some predictive genes using a suitable method. Self-organizing maps (SOM) (Kohonen, 1990), hierarchical clustering (Johnson, 1967), and consensus clustering (Swift *et al.*, 2004; Monti *et al.*, 2003) are traditional methods of clustering. There are two steps in clustering of data: determining the number of clusters and then clustering the data. Determination of optimal number of clusters for a training data is not a trivial task, and sometimes dynamic clustering method is applied.

4.2.1 Self-organizing map

SOM has a set of nodes with a simple topology (e.g., two dimensional grid) and a distance function $d(\mathbf{N}_1, \mathbf{N}_2)$ (usually Euclidean distance) on the nodes (Tamayo *et al.*, 1999). The Euclidean distance between two nodes/data points $\mathbf{N_1} = (x_1^{(N_1)}, x_2^{(N_1)}, \ldots, x_p^{(N_1)})$ and $\mathbf{N_2} = (x_1^{(N_2)}, x_2^{(N_2)}, \ldots, x_p^{(N_2)})$ is calculated as follows:

$$d(\mathbf{N}_1, \mathbf{N}_2) = \sqrt{\sum_{i=1}^{p} (x_i^{(N_1)} - x_i^{(N_2)})^2} \,. \tag{4.2.1}$$

Before clustering begins, all the data points (samples) of n-dimensional feature space are projected into a reduced dimensional work space through gene selection using a score metric. Then the clustering progresses in the following way:

1. Determine the number of clusters (nodes) m and randomly project the m nodes $(\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_m)$ on the two dimensional grid. That is, at initial, the coordinates $(x_1, x_2, \ldots, x_p)$ of the nodes are randomly chosen. Let the position of each node $\mathbf{N}_k (k = 1, 2, \ldots, m)$ at iteration t be $\mathbf{N}_k(t)$.

2. Randomly select a sample (data point) $\mathbf{x} = (x_1, x_2, \ldots, x_p)$ and find the nearest distance node $\mathbf{N}_c(t)$ and the neighborhood set $\mathbf{C}(t)$ (a set of indices of nearest neighbors) of $\mathbf{N}_c(t)$. Move the nodes towards $\mathbf{x}$ using the following formula:

$$\mathbf{N}_k(t+1) = \mathbf{N}_k(t) + \varphi_k(t)[\mathbf{x} - \mathbf{N}_k(t)] \tag{4.2.2}$$

where $\varphi_k(t)$ $(0 < \varphi_k(t) < 1; k = 1, 2, \ldots, m)$ is called adaptation gain. Different researchers have defined $\varphi_k(t)$ in different ways; however, in all cases, it is a decreasing function of iteration and eventually becomes zero at some iteration.

In GeneCluster (Tamayo *et al.*, 1999), adaptation gain is defined as follows:

$$\varphi_k(t) = \begin{cases} 0.02T/(T + 100t) & \text{if } k \in \mathbf{C}(t), \\ 0 & \text{otherwise;} \end{cases}$$

where T is the maximum number of iterations.

Kohonen (1990) has proposed the following function for $\varphi_k(t)$:

$$\varphi_k(t) = \begin{cases} \alpha(t) \exp(-||\mathbf{N}_k(t) - \mathbf{N}_c(t)||^2/\sigma^2(t)) & \text{if } k \in \mathbf{C}(t), \\ 0 & \text{otherwise;} \end{cases}$$

with $\alpha(t)$ and $\sigma(t)$ being decreasing functions of iteration and can be considered as the same. In (Smith and Ng, 2003), the authors have defined $\alpha(t)$ as follows: $\alpha(t) = \frac{\alpha(0)|\mathbf{C}(t)|}{\mathbf{C}(0)}$.

According to equation (4.2.2), the closest node $\mathbf{N}_c(t)$ is moved the most, whereas other nodes are moved by smaller amounts depending on their distance from $\mathbf{N}_c(t)$ in the initial geometry. The number of members in $\mathbf{C}(t)$ will be higher at the beginning and will decrease in successive phases, and finally would be zero.

3. Repeat step 2 for maximum number of iterations defined by the user. In GeneCluster, the process continues for 20,000–50,000 iterations.

4.2.2 Hierarchical clustering

Hierarchical clustering is widely used to build phylogenetic tree from biological data. The length of each branch represent the similarity between two objects.

Hierarchical clustering finds the pair of samples that are most similar, joins them together, and then identifies the next most similar pair of samples. This process continues until all the samples are joined into one giant cluster.

Before starting of the clustering process, we need to define a score metric for measurement of similarity between two samples. Since each sample is a vector of real-valued gene expressions, the Euclidean metric (minimum distance being the better) or the Pearson correlation coefficient metric (higher value being the better) can be used as a measure of similarity.

Once a table of similarity values between each pair of samples is built, the clustering process starts. The steps in a hierarchical clustering are as follows:

1. Assign each sample to its own cluster. Therefore, if there N samples in a data set, the total number of initial cluster will be N.

2. Find the most similar pair of clusters and merge them into a single pseudo-cluster, so that now you have one less cluster.

3. Remove the two clusters that were merged in the pseudo-cluster and calculate new distances (similarities) between the newly merged pseudo-cluster and each of the old clusters.

4. Repeat steps 2 and 3 until all items are clustered into a single cluster of size N.

There are three ways to calculate distance between two clusters in step 3: single-linkage, complete-linkage and average-linkage clustering. Let the two clusters be $\mathbf{C}_1$ and $\mathbf{C}_2$ and $sim(c_i, c_j)$ be the similarity score of the samples indexed by c_i and c_j. The similarity score between cluster $\mathbf{C}_1$ and $\mathbf{C}_2$ are calculated as follows:

- Single-linkage:

$$sim(\mathbf{C}_1, \mathbf{C}_2) = \max_{c_i \in \mathbf{C}_1, c_j \in \mathbf{C}_2} \{sim(c_i, c_j)\} .$$

- Complete-linkage:

$$sim(\mathbf{C}_1, \mathbf{C}_2) = \min_{c_i \in \mathbf{C}_1, c_j \in \mathbf{C}_2} \{sim(c_i, c_j)\} .$$

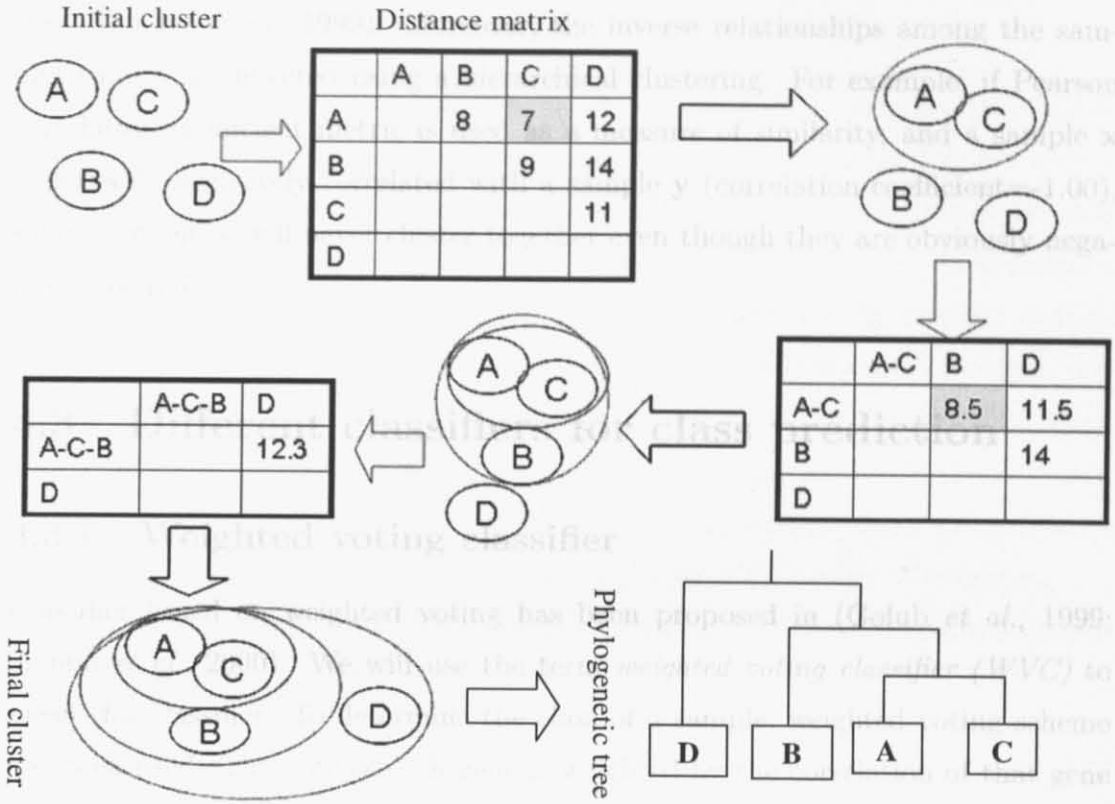


Figure 4.1: An example of hierarchical clustering of samples A, B, C, and D

- Average-linkage:

$$sim(\mathbf{C}_1, \mathbf{C}_2) = \text{avg}_{c_i \in \mathbf{C}_1, c_j \in \mathbf{C}_2}\{sim(c_i, c_j)\} \ .$$

An example of hierarchical clustering of four samples (A,B,C,D) using average linkage score metric is shown in Fig. 4.1.

Though hierarchical clustering has been a valuable method for class discovery, it has some shortcomings. Hierarchical clustering has been noted by statisticians to suffer from lack of robustness, nonuniqueness, and inversion problems that complicate the interpretation of the hierarchy. Finally, the deterministic nature of hierarchical clustering can cause points to be grouped based on local decisions, with no opportunity to reevaluate the clustering. It is known that the resulting trees can lock in accidental features, reflecting idiosyncrasies of the agglomeration

rule (Tamayo *et al.*, 1999). Moreover, the inverse relationships among the samples cannot be detected using a hierarchical clustering. For example, if Pearson correlation coefficient metric is used as a measure of similarity, and a sample **x** is perfectly negatively correlated with a sample **y** (correlation coefficient=-1.00), samples **x** and **y** will never cluster together even though they are obviously negatively related.

4.3 Different classifiers for class prediction

4.3.1 Weighted voting classifier

Classifier based on weighted voting has been proposed in (Golub *et al.*, 1999; Slonim *et al.*, 2000). We will use the term *weighted voting classifier (WVC)* to mean this classifier. To determine the class of a sample, weighted voting scheme has been used. The vote of each gene is weighted by the correlation of that gene with a particular class. The weight of a gene X_i is the signal-to-noise ratio defined as

$$W(X_i) = \frac{\mu_1 - \mu_2}{\sigma_1 + \sigma_2} \tag{4.3.1}$$

where μ_1, σ_1 and μ_2, σ_2 are the mean and standard deviation of the values of gene X_i in class 1 and 2, respectively. (Note that $W(X_i) = SNR(X_i)$.) The weighted vote of a gene X_i for an unknown sample $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is

$$V(X_i) = W(X_i)\left(x_i - \frac{\mu_1 + \mu_2}{2}\right) \tag{4.3.2}$$

where x_i is the value of gene X_i in that unknown sample **x**. Then, the class of the sample **x** is

$$class(\mathbf{x}) = sign\left\{\sum_{X_i \in G} V(X_i)\right\} \tag{4.3.3}$$

where G is the set of selected genes. If the computed value is positive, the sample **x** belongs to class 1; negative value means **x** belongs to class 2. This classifier is applicable to binary classification problems.

Prediction strength of WVC

It is always preferable for a classifier to give a confidence measure (prediction strength) of a decision about the class of a test sample. One can define a metric for decision confidence and determine empirically the probability that a decision of any particular confidence value is true according to that metric. By defining a minimum confidence level to classification, one can decrease the number of false positive and false negatives at the expense of increasing the number of unclassified samples. The combination of a good confidence metric and a good threshold value will result in a low false positive and/or low false negative rate without a concomitant high unclassified samples. The choice of appropriate decision confidence metric depends on the particular classifier and how the classifier is employed.

Golub *et al.* (1999) and Slonim *et al.* (2000) defined the prediction strength for weighted voting classifier as follows:

$$ps = \left| \frac{V_+ - V_-}{V_+ + V_-} \right| \tag{4.3.4}$$

where V_+ and V_- are respectively the absolute values of sum of all positive $V(X_i)$ and negative $V(X_i)$ calculated using equation (4.3.2).

The classification of an unknown sample is accepted if $ps > \theta$(θ is the prefixed prediction strength threshold), else the sample is classified as undetermined. In our experiment, we consider undetermined samples as misclassified samples.

4.3.2 Naive-Bayes classifier

Naive-Bayes classifier (NBC) uses probabilistic approach to assign the class to a sample. That is, it computes the conditional probabilities of different classes given the values of the genes and predicts the class with highest conditional probability. During calculation of conditional probability, it assumes the conditional independence of genes.

Let C denote a class from the set of m classes, $\{c_1, c_2, \ldots, c_m\}$, $\mathbf{X}$ is a sample described by a vector of n genes, i.e., $\mathbf{X} = (X_1, X_2, \ldots, X_n)$; the values of the genes are denoted by the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Naive-Bayes classifier tries

to compute the conditional probability $P(C = c_i|\mathbf{X} = \mathbf{x})$ (or in short $P(c_i|\mathbf{x})$) for all c_i and predicts the class for which this probability is the highest. Using Bayes' rule, we get

$$P(c_i|\mathbf{x}) = \frac{P(\mathbf{x}|c_i)P(c_i)}{P(\mathbf{x})} \ . \tag{4.3.5}$$

Since NB classifier assumes the conditional independence of genes, (4.3.5) can be rewritten as

$$P(c_i|\mathbf{x}) = \frac{P(x_1|c_i)P(x_2|c_i)\cdots P(x_n|c_i)P(c_i)}{P(x_1, x_2, \ldots, x_n)} \ . \tag{4.3.6}$$

The denominator in (4.3.6) can be neglected, since for a given sample, it is fixed and has no influence on the ranking of classes. Thus, the final conditional probability takes the following form:

$$P(c_i|\mathbf{x}) \propto P(x_1|c_i)P(x_2|c_i)\cdots P(x_n|c_i)P(c_i) \ . \tag{4.3.7}$$

Taking logarithm we get,

$$\ln P(c_i|\mathbf{x}) \propto \ln P(x_1|c_i) + \cdots + \ln P(x_n|c_i) + \ln P(c_i) \ . \tag{4.3.8}$$

For a symbolic (nominal) gene,

$$P(x_j|c_i) = \frac{\#(X_j = x_j, C = c_i)}{\#(C = c_i)} \tag{4.3.9}$$

where $\#(X_j = x_j, C = c_i)$ is the number of samples that belong to class c_i and gene X_j has the value of x_j, and $\#(C = c_i)$ is the number of samples that belong to class c_i. If a gene value does not occur given some classes, its conditional probability is set to $\frac{1}{2N}$, where N is the number of samples. For a continuous gene, the conditional density is defined as

$$P(x_j|c_i) = \frac{1}{\sqrt{2\pi}\sigma_{ji}} e^{-\frac{(x_j - \mu_{ji})^2}{2\sigma_{ji}^2}} \tag{4.3.10}$$

where μ_{ji} and σ_{ji} are the expected value and standard deviation of gene X_j in class c_i. Taking logarithm of equation (4.3.10) we get,

$$\ln P(x_j|c_i) = -\frac{1}{2}\ln(2\pi) - \ln\sigma_{ji} - \frac{1}{2}\left(\frac{x_j - \mu_{ji}}{\sigma_{ji}}\right)^2 \tag{4.3.11}$$

Since the first term in (4.3.11) is constant, it can be neglected during calculation of $\ln P(c_i|\mathbf{x})$.

The advantage of the Naive-Bayes classifier is that it is simple and can be applied to multi-class classification problems.

Prediction strength of NBC

For Naive-Bayes classifier, the prediction strength metric for two class problems can be defined as the relative log likelihood difference of the winner class from the loser class (Keller *et al.*, 2000b). That is, the prediction strength of the classifier for an unknown sample $\mathbf{x}$ is

$$ps = \frac{\ln P(c_{winner}|\mathbf{x}) - \ln P(c_{loser}|\mathbf{x})}{\ln P(c_{winner}|\mathbf{x}) + \ln P(c_{loser}|\mathbf{x})} . \tag{4.3.12}$$

4.3.3 Decision tree: C4.5

The C4.5 (Quinlan, 1993) represents a classification model by a decision tree. It is run with the default values of its parameters. The tree is designed in a top-down way, dividing the training set and beginning with the selection of the best variable at the root of the tree. The best gene is selected by maximizing a splitting criterion based on information theoretic approach.

For each gene in a microarray data set, a threshold, that maximizes the splitting criteria, is determined by scoring the number of cases of the data set on the value of the gene; every pair of values suggest a threshold in their midpoint, and the threshold that yields the best value of splitting criterion is selected. A descendant of the root node is then created for each possible value of the selected gene, and the training samples are associated with the appropriate descendant node. The entire process is recursively repeated using the training samples associated with each descendant node.

The process stops when all the samples at each node of the tree belong to the same category or the best split of the node does not surpass a fixed chi-square significant threshold. Then, the tree is simplified by a pruning mechanism to avoid

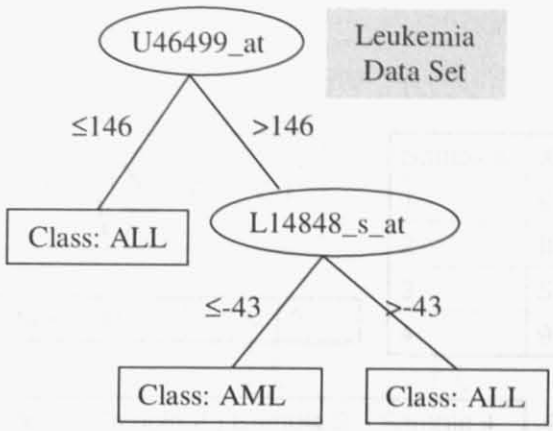


Figure 4.2: An example of classification by decision tree

overspecialization. An example of classification by decision tree (taken from (Inza *et al.*, 2002)) is shown in figure 4.2.

4.3.4 k-Nearest neighbor classifier

The k-nearest neighbor (kNN) classifier (Dasarathy, 1991), an extension of nearest neighbor classifier (IB1)(Aha *et al.*, 1991), has long been used by the pattern recognition and machine learning communities in supervised classification tasks. The basic approach involves storing all the training samples, when a test sample is presented, retrieving k training samples that are nearest to this test sample and prediction of the label of the test sample by majority voting. The distance between two samples $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ is calculated as follows:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} w_i (x_i - y_i)^2} \tag{4.3.13}$$

where n is the number of genes in the data set and w_i is the weight of gene i. In our experiments, we set $w_i = 1$, and the distance between two samples becomes Euclidian distance. To avoid a tie, the value of k should be an odd number (and of course, $k <$ number_of_training_samples) for binary classification. Examples of IB1 and kNN classifiers are given in Figs. 4.3 and 4.4.

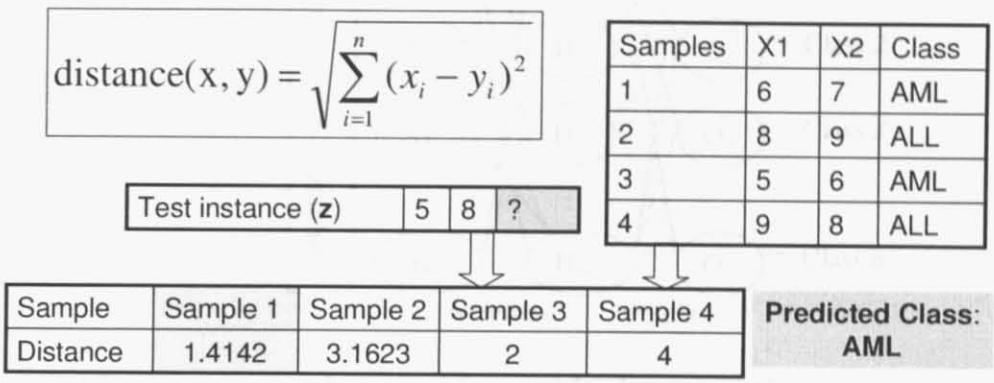


Figure 4.3: Class prediction by nearest neighbor classifier (IB1)

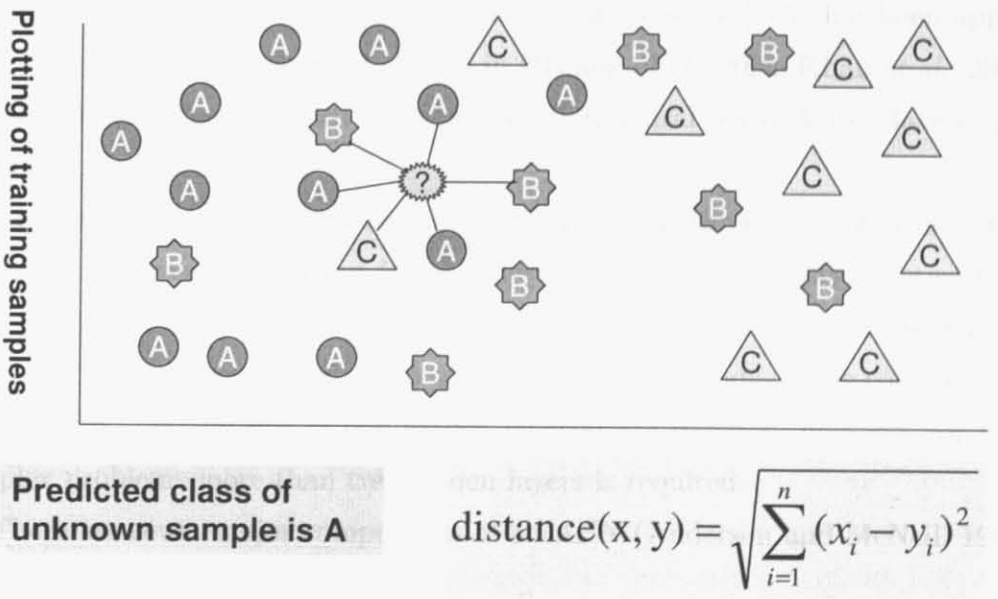


Figure 4.4: Class prediction by k-nearest neighbor (kNN) classifier

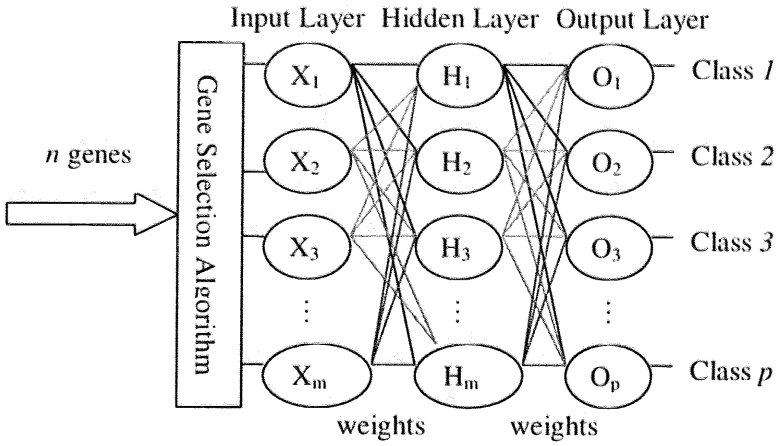


Figure 4.5: Classification by a feed forward back propagation neural network

kNN is very easy to implement and can provide good classification accuracy if the features are chosen and weighted carefully in computation of distance.

4.3.5 Artificial neural network

Artificial neural network (ANN) or simple neural network (NN) has been applied in classification of gene expression data in (Hwang *et al.*, 2002; Khan *et al.*, 2001). Fig. 4.5 shows a feed forward back propagation neural network for classification of gene expression data.

An artificial neural network consists of the following three types of layers: input layer, hidden layer, and output layer. The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. However, the number of hidden layers required to solve a problem depends on the complexity of the problem. Usually two or one hidden layer(s) are required; however, for some complex problems more than two hidden layers is required.

There are seven major components of an ANN (Anderson and McNeil, 1992):

1. **Weighting factors:** A neuron usually receives many simultaneous inputs. Each input has its own relative weight which gives the input the impact that it needs on the processing element's summation function.

2. **Summation function:** The first step in a processing element's operation

is to compute the weighted sum of all of the inputs.The total input signal is the dot product of the input and weight vectors.

3. **Transfer function:** The result of the summation function, almost always the weighted sum, is transformed to a working output through the transfer function. In the transfer function the summation total can be compared with some threshold to determine the neural output. The threshold, or transfer function, is generally non-linear. Typically the sigmoid function is used as the transfer function. The sigmoid function is defined as follows:

$$Y_j = \frac{1}{1 + e^{-\Sigma_j}} \tag{4.3.14}$$

where Σ_j is output of the summation function and Y_j is the output activity of node j. Prior to applying the transfer function, uniformly distributed random noise may be added. The source and amount of this noise is determined by the learning mode of a given network paradigm.

4. **Scaling and limiting:** The result of the processing element's transfer function can pass through additional processes that scale and limit the transfer value. Scaling simply multiplies the transfer value by a scale factor, and then adds an offset. Limiting is the mechanism that insures that the scaled result does not exceed an upper or lower bound. This limiting is in addition to the hard limits that the original transfer function may have performed.

5. **Output function**: Each processing element is allowed one output signal which it may output to hundreds of other neurons. Normally, the output is directly equivalent to the transfer function's result. Some network topologies, however, modify the transfer result to incorporate competition among neighboring processing elements. Neurons are allowed to compete with each other, inhibiting processing elements unless they have great strength. Competition can occur at one or both of two levels. First, competition determines which artificial neuron will be active, or provides an output. Second, competitive inputs help determine which processing element will participate in the learning or adaptation process.

6. **Error function and back-propagated value:** In most learning networks the difference between the current output and the desired output is calculated. This raw error is then transformed by the error function to match a particular network architecture. The most basic architectures use this error directly, but some square the error while retaining its sign, some cube the error, other paradigms modify the raw error to fit their specific purposes. The artificial neuron's error is then typically propagated into the learning function of another processing element. This error term is sometimes called the current error.

   The current error is typically propagated backwards to a previous layer. Yet, this back-propagated value can be either the current error, the current error scaled in some manner (often by the derivative of the transfer function), or some other desired output depending on the network type. Normally, this back-propagated value, after being scaled by the learning function, is multiplied by the incoming connection weights to modify them before the next learning cycle.

7. **Learning function:** The purpose of the learning function is to modify the different connection weights on the inputs of each processing element according to some neural based algorithm. This process of changing the weights of the input connections to achieve some desired result can also be called the adaption function, as well as the learning mode.

In gene expression based classification using an ANN, the data corresponding to the selected genes of a gene subset are presented to the input layer, and the number of input nodes is equal to the number of selected genes. The number of output layers depends on the problem. For a binary problem, only one output layer and one slice point (threshold) (α) is sufficient for the prediction of the labels. If the output of the neural network is greater than or equal to α, the predicted class is one type; otherwise, the predicted class is other type. However, for a multiclass problem, usually the number of output layers and the number of slice points are equal to the number of types of samples in the data set. The determination of

optimal slice points for a multiclass problem is not an easy task; sometime, these points are determined through some learning process.

4.3.6 Support vector machine

Support vector machine (SVM) (Vapnik, 1995) in binary classification maximizes the distance between a hyperplane $\mathbf{w}$ and the closest samples from the hyperplane. The hyperplane found by the SVM in feature space corresponds to a non-linear decision boundary in input space.

Let us assume that we have l training samples and the class of each training vector $\mathbf{x_i} = (x_{i_1}, x_{i_2}, \ldots, x_{i_n})$ $(i = 1, 2, \ldots, l)$ is given by $y_i \in \{-1, +1\}$. Given a new test sample $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ to classify, its label is assigned according to the following decision function:

$$D(\mathbf{x}) = sign\left(\sum_{i=1}^{l} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b\right) \tag{4.3.15}$$

where $K(\mathbf{x_i}, \mathbf{x}) \equiv \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$ is called kernel function, α_i $(0 \leq \alpha_i \leq C$; C: cost) is the weight of $\mathbf{x_i}$ and b is a biased value. Some widely used kernel functions are: polynomial kernel $[K(\mathbf{x}_i, \mathbf{x}_j) = (a\mathbf{x}_i^T \mathbf{x}_j + r)^d]$, RBF (Radial Basis Function) kernel $[K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma \|\mathbf{x}_i - \mathbf{x_j}\|^2}]$, and linear kernel $[K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j]$. The value of α_i is found by minimizing the following equation:

$$F(\alpha) = \frac{1}{2} \sum_{i=1, j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x_i}, \mathbf{x_j}) - \sum_{i=1}^{l} \alpha_i \tag{4.3.16}$$

subject to

$$\sum_{i=1}^{l} \alpha_i y_i = 0 \ . \tag{4.3.17}$$

The maximum margin hyperplane is given by

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x_i}) \ . \tag{4.3.18}$$

An example of optimal hyper-plane for SVM is shown in Fig. 4.6.

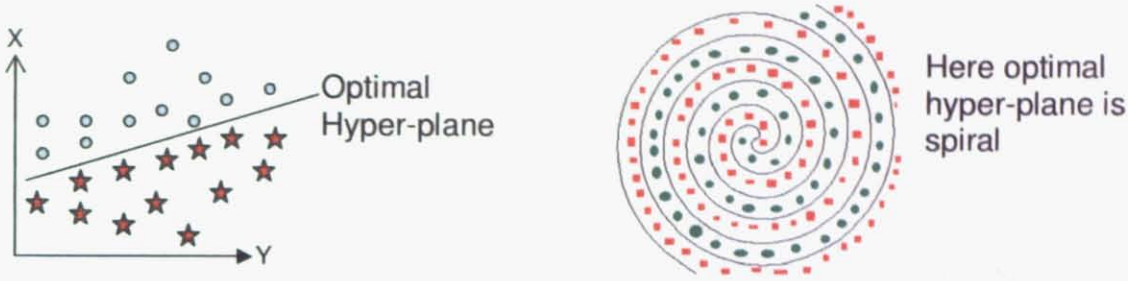


Figure 4.6: Examples of hyper-planes in support vector machine

After successful training of SVM, most α_i's are zero, and the training patterns with non-zero weights are called support vector, and those with strict inequality $0 < \alpha_i < C$ are called marginal support vectors. Many information about SVM can be found at `http://www.kernel-machines.org`.

Though the original SVM was intended for binary classification, it has been extended to multiclass classification using 'one-vs-other' and 'one-vs-one' (all pairs) methods. We will describe only the second method ('one-vs-one') of SVM for multiclass classification due its use in our experiments. In 'one-vs-one' method (Kreßel, 1999), $c(c-1)/2$ classifiers (c= number of classes) are constructed, where each classifier is trained on data from two classes (i, j), and the class of a test sample $\mathbf{x}$ is predicted by 'winner-takes-all' voting strategy. If the decision function says that $\mathbf{x}$ is in class class i, the vote for the ith class is increased by one, else the vote for jth class is increased by one. Then $\mathbf{x}$ is predicted to be in the class that has the highest votes. In the case that two classes have identical votes, the one with lower index is selected.

The SVM experiments used in this dissertation are performed by using an implementation of LIBSVM (Chang and Lin, 2001) (software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`).

4.4 Accuracy estimation through cross-validation

When the number of samples in a training data set is small, cross-validation technique is applied to measure the goodness of a gene subset using a classifier. In

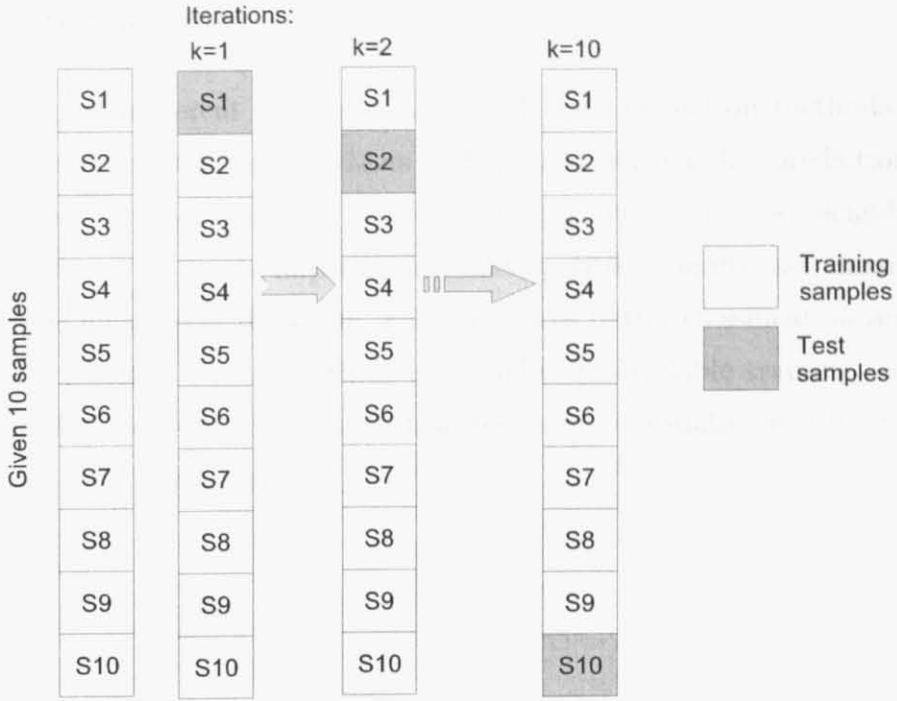


Figure 4.7: An example of leave-one-out-cross-validation technique for accuracy estimation

k-fold cross-validation, the data D is randomly partitioned into k mutually exclusive subsets, $D_1, D_2, \ldots, D_k$ of approximately equal size. The classifier is trained and tested k times; each time $i(i = 1, 2, \ldots, k)$, it is trained with $D \backslash D_i$ and tested on D_i. When k is equal to the number of samples in the data set, it is called leave-one-out-cross-validation (LOOCV) (Kohavi, 1995). The cross-validation accuracy is the overall number of correctly classified samples, divided by the number of samples in the data. When a classifier is stable for a given data set under k-fold cross-validation, the variance of the estimated accuracy would be approximately equal to $\frac{a(1-a)}{N}$ (Kohavi, 1995), where a is the accuracy and N is the number of samples in the data set. An example of leave-one-out-cross-validation technique for accuracy estimation is given in Fig. 4.7.

4.5 Summary

In this chapter, different class discovery and class prediction methods are described. For class discovery, SOM is widely used whereas for prediction of the label of a test sample, the most widely used classifier is k-nearest neighbor classifier since it can be easily implemented and applied to multiclass classification, and on some microarray data sets, kNN produces better classification accuracies than support vector machine. Since the number of available training samples is very small, leave-one-out-cross-validation technique is usually used to evaluate a gene subset with a classifier.

Chapter 5

Gene Selection by Random Probabilistic Model Building Genetic Algorithm

5.1 Introduction

In this chapter, we propose a new adaptive search method to extract informative genes from microarray data.We call our gene selection method **random probabilistic model building genetic algorithm (RPMBGA)**. Our method belongs to the category probabilistic model building genetic algorithm (PMBGA) (Pelikan *et al.*, 1999), which is a variant of genetic algorithm (GA). Instead of applying crossover and mutation operators, a PMBGA generates new possible solutions (individuals) by sampling the probability distribution that is calculated from the selected solutions of previous generations. Different PMBGAs assume different structures of variables and calculate probability distribution accordingly. A good review on PMBGAs (also known as estimation of distribution algorithms (Mühlenbein and Paaß, 1996)) can be found in (Larrañaga and Lozano, 2001; Paul and Iba, 2003a,b). PMBGA has been proved to be an efficient method for solving certain kinds of problems that are very hard for ordinary genetic algorithms.

5.2 Motivation

The success of a traditional genetic algorithm depends on the appropriate choice of crossover and mutation operators; similarly, the success of a PMBGA depends

on its capability of learning a structure of the variables from the selected individuals. The structure of genes of a microarray data set can be described by a Bayesian network. But learning of a Bayesian network from data is an NP-hard problem. For medium size problems, a Bayesian network of variables is built from the selected individuals by using some kinds of greedy algorithms, which use either bayesian information criterion (BIC)(Schwarz, 1978) or bayesian dirichlet equivalence (BDe)(Geiger and Heckerman, 1994) metric to measure the goodness of a structure. For the problems containing thousands of variables, it would be virtually impossible to build a network structure from data.

On the other hand, if the recombination operators (especially mutation operator) of a GA are not carefully designed, it would be very difficult to generate compact size gene subsets and take much time to calculate classification accuracies of bigger size gene subsets. These have motivated us to design RPMBGA that successively reduces the number of genes of different individuals but keeps diversity in the population of individuals in successive generations. The details of RPMBGA are given below.

5.3 Details of RPMBGA

5.3.1 Notations

Before the details of RPMBGA, let us give some notations we use in this chapter. We use the term *individual* or *gene subset* to mean a possible solution of the problem. If there are n genes in a microarray data set, each possible solution would be an n-bit binary string. In a binary string, the genes at the positions marked with a 1 are included in that gene subset. Suppose, the random variable $X_i \in \{0, 1\}$ corresponds to gene i, and x_i is the value of X_i. So, a solution of the problem is represented as $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. $p(x_i, t)$ is the probability of X_i being 1 at generation t, and $q(x_i, t)$ is the marginal distribution of X_i across the selected individuals at generation t. N is the number individuals in a population, M is the number of individuals selected from a population, and x_i^j is the value of

the variable X_i in individual j.

5.3.2 Initial population generation

In our algorithm, whether a gene would be selected or not depends on its probability $p(x_i, t)$. Initial population of different gene subsets is generated by setting the probability $p(x_i, t)$ to 0.5 and applying the following decision rule:

$$x_i^j = \begin{cases} 1 & \text{if } r < p(x_i, t); \\ 0 & \text{otherwise} \end{cases} \tag{5.3.1}$$

where $r \in [0, 1]$ is a random number usually generated by calling the *rand*() function of a programming language. Let us give an example of generating initial population of four genes in detailed. Given the initial probability vector $p(\mathbf{x}, 0) = (0.5, 0.5, 0.5, 0.5)$, N random vectors are generated. Suppose two of them are $R_1 = (0.002, 0.69, 0.045, 0.85)$ and $R_2 = (0.73, 0.032, 0.45, 0.21)$. By using decision rule (5.3.1), we get two gene subsets as $(1, 0, 1, 0)$ and $(0, 1, 1, 1)$.

5.3.3 Generation of new solutions (offspring)

After generation of initial population, we need to update the probability vector to produce new solutions. In PBIL (Baluja, 1994), a member of the group PMBGA, the probability of a variable X_i is updated by the weighted average of $p(x_i, t)$ and the marginal distribution of that variable $q(x_i, t)$:

$$p(x_i, t + 1) = \alpha p(x_i, t) + (1 - \alpha) q(x_i, t) \tag{5.3.2}$$

where $\alpha \in [0, 1]$ is called learning rate that is usually fixed at a value during initialization. In a data set containing smaller number of genes, PBIL may produce good results but in microarray data sets containing huge number of genes, it may not return compact size gene subsets for a fixed value of α. We performed experiments on different microarray data sets with different values of α but in each run, it terminated with many genes selected. In the research on microarray data, it is assumed that only a few genes anticipate the pathological behavior of cancers.

Smaller number of genes will be selected if we can somehow reduce the probability of a gene being selected and can keep the search adaptive. Though theoretical analysis of our method would not be provided in this chapter, we achieve this goal by incorporating a random variable in (5.3.2). So, we update probability as follows:

$$p(x_i, t+1) = \alpha\beta_i p(x_i, t) + (1-\alpha)(1-\beta_i)q(x_i, t) \tag{5.3.3}$$

where $\beta_i \in [0, 1]$ is a random number. For a fixed value of α, $p(x_i, t+1)_{PBIL} \geq p(x_i, t+1)_{RPMBGA}$. Therefore, our method will select smaller number of genes as compared to PBIL. The marginal distribution of X_i is calculated as follows:

$$q(x_i, t) = \frac{\sum_{j=1}^{M} x_i^j}{M} \tag{5.3.4}$$

where x_i^j is the value of the variable X_i in individual j.

5.3.4 Evaluation of a gene subset

A gene subset (individual) is evaluated by its accuracy on the training data and the number of genes selected in it. Usually, the value of the fitness function is used as an evaluation measure. In our method, we calculate the fitness of an individual as follows:

$$fitness(\mathbf{x}) = w * A(\mathbf{x}) + (1-w) * (1 - NGS(\mathbf{x})/n) \tag{5.3.5}$$

where $A(\mathbf{x}) \in [0, 1]$ is the accuracy on training data using only the expression values of the selected genes in $\mathbf{x}$, $NGS(\mathbf{x})$ is the number of genes selected in $\mathbf{x}$ and $w \in [0, 1]$ is the assigned weight of accuracy. In (5.3.5), we have scalarized the two objectives of gene identification task into one. In our experiments, we give more emphasis on accuracy rather than on number of selected genes. Hence in all our experiments, $w > (1-w)$.

5.3.5 Overall gene selection procedure

The steps in our gene selection algorithm are: generation and evaluation of initial population, selection of some promising individuals for calculation of marginal

probabilities, generation and evaluation of offspring, and creation of new population by combining old population and new offspring. The overall procedure is as follows:

PROCEDURE RPMBGA;
  Generate initial population of different gene subsets;
  Evaluate initial population using (5.3.5);
  WHILE (*termination_criteria* NOT *satisfied*) **DO**
    Select M promising individuals;
    Calculate marginal distribution using (5.3.4);
    Update probability vector according to (5.3.3);
    FOR i=1 to Q **DO** //Q=number of offspring to generate
      FOR j=1 to n **DO**
        r=rand();
        Generate x_j^i using decision rule (5.3.1);
    Evaluate the newly generated gene subsets using (5.3.5);
    Create new population by combining old and new gene subsets;

5.3.6 Example of offspring generation by RPMBGA

Let us give an example of generation of new offspring in RPMBGA containing five genes.

1. Suppose the initial probability vector is given as

$$p(\mathbf{x}, 0) = (0.5, 0.5, 0.5, 0.5, 0.5) ,$$

   and the initial population contains the following individuals (fitness follows colon):
   (a) 10011:0.59 (b) 11010:0.60 (c) 10001:0.85 (d) 01110:0.75 (e) 00111:0.54.

2. Select some individuals based on fitness (b,c,d):
   11010:0.60, 10001:0.85 and 01110:0.75.

3. Calculate marginal distribution of each X_i:

$$q(\mathbf{x}, 0) = (\frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{2}{3}, \frac{1}{3}).$$

4. Generate a random vector:

$$\beta = (0.10, 0.25, 0.43, 0.67, 0.90).$$

5. Update the probability vector using (5.3.3)($\alpha = 0.9$):

$$p(\mathbf{x}, 1) = (0.1050, 0.1625, 0.2125, 0.3235, 0.4083).$$

6. Generate a set of random vectors:

   (a) $R_1 = (0.10, 0.054, 0.7, 0.8, 0.77)$,

   (b) $R_2 = (0.23, 0.56, 0.20, 0.15, 0.95)$,

   (c) $R_3 = (0.45, 0.054, 0.17, 0.53, 0.57)$.

7. Generate new offspring by comparing $p(\mathbf{x}, 1)$ and each random vector $R_i (i = 1, 2, 3)$ and applying decision rule (5.3.1):
   (a)11000 (b)00110 (c)01100.

8. Evaluate new offspring, and generate new population by combining old population and new offspring.

5.4 Evaluations of RPMBGA on microarray datasets

5.4.1 Microarray datasets

To evaluate the accuracy of our proposed method, we chose three microarray data sets of cancer research. The data sets include lung carcinoma (Bhattacharjee *et al.*,

Table 5.1: Microarray data sets used in the experiments

Data Set	#Genes	#Classes	#Samples
Lung carcinoma	3312	5	203
Brain cancer	4434	2	50
Prostate cancer	5966	2	102

2001), brain cancer (Nutt *et al.*, 2003) and prostate cancer (Singh *et al.*, 2002). Summary of the data sets are shown in Table 5.1. In the table, #Genes denotes the number of genes that were left after preprocessing. After preprocessing of the data sets (see Chap. 2), the values of each gene across different samples were linearly normalized in [0,1]. The lung carcinoma and the prostate cancer data sets were divided into mutually exclusive training and test subsets containing (103,100), and (51,51) samples, respectively (ratio is 1:1) whereas the 21 classic samples of brain cancer were used as training data and the remaining 29 non-classic samples as test data. This split of each data set into training and test subsets remained the same for all the experiments.

5.4.2 Experimental setup

We generated initial population randomly with the probability of each gene being selected was 0.5 (equal probability of being selected or not). The settings of the parameters of gene selection algorithm were: population size=100, offspring size=100, maximum number of generation=100, total run=20, $w = 0.75$ and $\alpha = 0.1$. The value of w was chosen to give more emphasis on accuracy rather than on the number of selected genes because the ultimate objective of this research is the accurate classification of patient samples. Our replacement strategy was CHC (Eshelman, 1991) in which we combined the old population and the newly generated offspring and then selected the best 100 individuals for the next generation. For calculation of marginal probabilities, we selected the best half of the population.

We used support vector machine (SVM) and k-nearest neighbor (kNN) as classifiers. SVM is well suited to the analysis of broad patterns of gene expressions

from DNA microarray data. It can easily deal with a large number of genes with a smaller number of training patterns. kNN is easy to implement and widely used in analysis of gene expression data. For SVM, we used RBF kernel with values of C=32, and $\gamma = 0.0078125$; these values were obtained by applying grid search on the training data as recommended in (Chang and Lin, 2001). In this chapter, we report the experimental results of kNN with $k = 11$ because by performing different experiments with $k = 10$ and 11, we found that kNN with $k = 11$ had produced the better results on the data sets. Our gene selection algorithm terminated when there was no improvement of the fitness value of the best individual in the population in 10 consecutive generations or maximum number of generations had passed. After termination of the algorithm, instead of taking the best one that had the highest fitness value, we took all the gene subsets from the population that had the best training accuracy (fitness might be different) and calculated test accuracy of each gene subset by using either SVM or kNN, whatever might be the case. That is why, the number of gene subsets selected by our method in 20 runs were greater than or equal to 20.

5.4.3 Results

The experimental results presented in this section pursue two objectives. The first objective is to show that gene selection is needed for better classification of microarray data while the second objective is to show that adaptive searches like RPMBGA select highly discriminative genes than SNR does. To achieve these objectives, we performed six types of experiments. In the first four experiments, we calculated the accuracies of single genes, all genes, and the genes selected by RPMBGA and SNR on the three data sets. Then, we calculated the training and test accuracies of the genes selected with RPMBGA and SNR. To denote a gene, we use feature# (probe_set#) of that gene in the microarray data set.

Table 5.2: Overall accuracies by single genes on each data set

Data Set	SVM	kNN
Lung carcinoma	34842_at (77.34, 76.85)	41325_at (72.91, 80.79)
	36160_s_at (77.34, 77.83)	33322_i_at (68.47, 80.30)
	40825_at (77.34, 78.32)	32321_at (68.47, 79.80)
Brain cancer	36617_at (72.0, 74.0)	630_at (62.0, 84.0)
	40367_at (72.0, 64.0)	35163_at (68.0, 82.0)
	1113_at (70.0, 68.0)	41753_at (56.0, 82.0)
Prostate cancer	37639_at (84.31, 83.33)	32598_at (62.75,88.23)
	37720_at (84.31, 80.39)	40856_at (66.67, 84.31)
	34840_at (79.41, 76.47)	1767_s_at (71.57, 83.33)

Overall accuracies of single genes and all genes

Before implementation of our gene selection method on the data sets, we applied both SVM and kNN classifier to the data containing single genes' expression values to determine whether a single gene exists that can classify all the (training+test) samples without any error. Our findings are summarized in Table 5.2. In the table, we have reported the top 3 genes that produced the higher overall accuracies. For each gene, the accuracies found by SVM and kNN are written in the parenthesis; the kNN accuracy follows the comma in the parenthesis. For each data set, the top three genes selected by SVM were different from those by kNN. For a single gene, the accuracy by SVM was also different from the accuracy by kNN. None of the genes produced 100% classification accuracy on any data set. The best overall accuracies using SVM were 77.34%, 72% and 84.31%, respectively on lung carcinoma, brain cancer and prostate cancer data sets; using kNN as a classifier, the best overall accuracies found on those data sets were 80.79%, 84.0% and 88.23%, respectively.

Since we did not find any single gene that can classify all the samples 100% accurately, we next calculated the overall accuracy on those samples using the expression values of all genes. The results are shown in Table 5.3. For each data set, the best accuracy was obtained by applying SVM but that was less than 100%. Then we investigated whether there could be found gene subsets that would

Table 5.3: Overall accuracy by all genes on each data set

Data Set	SVM	kNN
Lung carcinoma	95.07	93.10
Brain cancer	84.0	80.0
Prostate cancer	89.22	84.31

produce better classification accuracies on these data sets or not.

Overall accuracies of the genes selected with RPMBGA and SNR

Next we applied our gene selection algorithm RPMBGA on all the samples. The best results are presented in Table 5.4 while the average results are reported in Table 5.5. In Table 5.5, a value of the form $a \pm b$ represents average value a with standard deviation b. From these tables, we find that whatever classifier is used, the best as well as the average overall accuracy of RPMBGA is much better than the accuracy of either a single gene or all genes. Moreover, a smaller size gene subset that results in higher classification accuracy may provide more insights into molecular classification and diagnosis of cancers. This suggests that we need gene selection.

Since SNR is widely used to identify discriminative genes from microarray data, we performed experiments using different number of genes selected by SNR to determine whether our method is superior to SNR or not. On brain and prostate cancers, we performed 20 experiments using different number of genes (5,10,15,...,100). Similarly, we performed 20 experiments on lung carcinoma data using 10, 20, ..., 200 genes. Moreover, we performed experiments using the same number of genes as of RPMBGA that produced the best overall accuracies using SVM and kNN. In Table 5.6, we report those results. The 90-gene subset, selected by SNR, produced the best 97.04% and 97.54% overall accuracies on the lung carcinoma data using SVM and kNN, respectively. On the brain cancer data, the 30- and 75-gene subsets produced the best 88% and 92.0% overall accuracies using SVM and kNN, respectively. Similarly, the best accuracy obtained on prostate

Table 5.4: Best results of RPMBGA on all samples

Data set	Overall Accuracy		#Genes	
	SVM	kNN	SVM	kNN
Lung carcinoma	98.03	95.56	67	29
Brain cancer	96.0	98.0	4	10
Prostate cancer	98.04	99.02	17	4

Table 5.5: Average results of RPMBGA on all samples

Data set	Metric	SVM	kNN
Lung carcinoma	Overall accuracy	97.30±0.44	94.28 ±0.53
	#Genes	73.09±21.83	39.0±13.68
Brain cancer	Overall accuracy	93.17±2.02	94.40±0.90
	#Genes	20.55±7.51	10.91±2.15
Prostate cancer	Overall accuracy	96.62±0.62	96.81±1.10
	#Genes	48.52±47.07	6.37±4.20

cancer data was 95.10% using either a 17-gene subset with SVM or a 5-gene subset with kNN. All the best overall accuracies of RPMBGA were superior to the accuracies of the gene subsets found by SNR with corresponding classifier, except the best overall accuracy of kNN on lung carcinoma. Even the average accuracy of the gene subsets of RPMBGA on each data set, except kNN accuracy on lung carcinoma data, was superior to the best accuracy of the gene subset selected by SNR. Since all the best overall accuracies of RPMBGA, except kNN accuracy on lung carcinoma, are better than the accuracy of the single genes, all genes or the best gene subset found by SNR, we can claim that RPMBGA can be used to select highly discriminant genes for classification of tumor samples.

Table 5.6: Overall accuracy by the genes selected by SNR

Lung carcinoma			Brain cancer			Prostate cancer		
#Genes	SVM	kNN	#Genes	SVM	kNN	#Genes	SVM	kNN
29	92.61	93.60	4	78.0	86.0	4	90.2	94.11
60	96.55	97.04	10	84.0	90.0	5	91.18	95.10
67	96.06	96.06	30	88.0	86.0	17	95.10	93.14
90	97.04	97.54	75	88.0	92.0	20	95.10	93.14

Interestingly, among the members of the best gene subsets found by SNR with SVM, and those by RPMBGA with SVM, all the top three single genes of a data set found by SVM are included in the best gene subset found by SNR, but none of them appear in the best gene subset found by RPMBGA. From this, we can infer that there exist some kinds of correlations among the selected genes of the best subset; when we take a single gene from the subset, the correlation breaks down and it does not produce good accuracy on the data set. SNR fails to select an informative gene subset because it does not consider the interactions among the genes, rather it selects genes based on single genes' capability of data separation. On the contrary, RPMBGA that generates subsets of more than one genes a time, preserves the interactions among the genes that result in good classification accuracy.

Training and test accuracies of the genes selected with RPMBGA and SNR

We performed experiments using RPMBGA and SNR with either of the classifiers on each data set divided into mutually exclusive training and test data. During selection of the best gene subset, only training data were used. The best and average results of RPMBGA on training and test data are presented in tables 5.7 and 5.8. Here we use the notation (a, b) (if needed and not otherwise stated) to denote training accuracy a and test accuracy b. SVM obtained better accuracies than kNN on lung carcinoma data. However, the accuracies of the two classifiers on the other two data sets were almost similar.

On lung carcinoma data, we obtained 48 and 49 gene subsets in 20 runs by using RPMBGA with SVM and kNN, respectively. Out of these gene subsets, the best training and test accuracies by SVM and kNN were found by 107- and 44-gene subsets, and the accuracies were (97.09%, 98.0%) and (95.15%, 92.0%), respectively. The lowest training and test accuracies by SVM and kNN were (96.12%, 87.0%) and (91.26%, 85%), which were obtained by the gene subsets having respectively 63 and 86 genes.

Applying RPMBGA with SVM on brain cancer data, we got 1971 gene subsets in 20 runs each having 100% training accuracy on 21 training samples; however, the test accuracy on the 29 test samples was in the range [72.41%, 48.28%]. Using kNN as the classifier, we got 859 genes that produced 100% training accuracy, and test accuracy in the range [75.86%, 37.93%]. The descriptions of the selected genes of the best subset, obtained by applying RPMBGA with SVM, are given in Table 5.10.

By running our method with SVM on prostate cancer data, we found 76 gene subsets that resulted in 100% training accuracy; the test accuracy of these gene subsets was in the range [98.04%, 78.43%]. Using kNN as classifier, we did not find any gene subset that produced 100% training accuracy. The highest 98.04% training accuracy was obtained by 67 gene subsets, and the highest and the lowest test accuracies of these subsets were 98.04% and 82.34%, respectively. The descriptions of the selected genes in the best subset, obtained by applying RPMBGA with SVM, are given in Table 5.11.

We further investigated the performance of SNR on three data sets when they were divided into training and test sets. Our findings are summarized in Table 5.9. The best accuracies by SVM on the lung carcinoma, brain cancer and prostate cancer were: (96.12%, 97.0%), (100.0%, 48.28%) and (96.08%, 94.12%) using the gene subsets of 160, 20 and 55 genes, respectively. Using kNN as a classifier, the best accuracies observed on those data were: (92.23%, 92.0%), (100%, 62.07%) and (96.08%, 96.08%) using the 30, 40 and 5-gene subsets, respectively. The accuracies corresponding to our best gene subsets are also provided in the table. Again, our method outperforms SNR in terms of acquired classification accuracy.

In figures 5.1, 5.2 and 5.3, we have plotted the training accuracy vs test accuracy of different gene subsets selected with RPMBGA and SNR using SVM. Since the accuracies corresponding to different gene subsets are not monotonic, we have shown the graphs as dot plots. We have provided these graphs to show the performance of RPMBGA and SNR in selection of predictive genes.

Table 5.7: Best results of RPMBGA on training and test samples

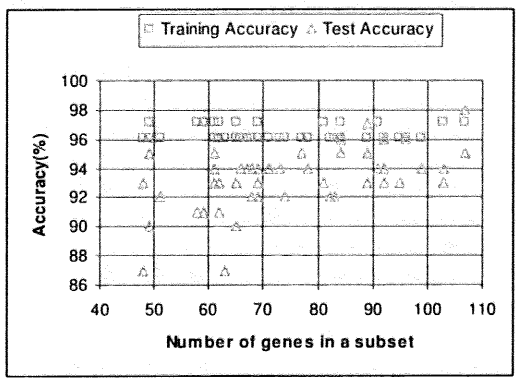
Data set	Classifier	Best accuracy		#Genes
		Training	Test	
Lung	SVM	97.09	98.0	107
carcinoma	kNN	95.15	92.0	44
Brain	SVM	100.0	72.41	9
cancer	kNN	100.0	75.86	8
Prostate	SVM	100.0	98.04	10
cancer	kNN	98.04	98.04	4

Table 5.8: Average results of RPMBGA on training and test samples

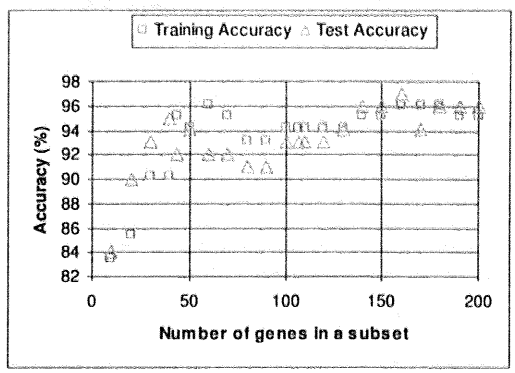
Data set	Classifier	Average accuracy		#Genes
		Training	Test	
Lung	SVM	96.45±0.47	93.37±2.13	76.35±17.04
carcinoma	kNN	92.15±0.74	90.06±1.72	35.61±15.51
Brain	SVM	100.0±0	52.82±4.41	8.95±2.18
cancer	kNN	99.35±1.63	52.26±5.13	8.48±0.75
Prostate	SVM	99.06±0.98	87.86±3.97	17.50±19.37
cancer	kNN	97.16±1.00	87.43±6.04	5.98±4.59

Table 5.9: Training and test accuracies of the genes selected by SNR. For each gene subset, first row contains training accuracy while second row contains test accuracy

Lung carcinoma			Brain cancer			Prostate cancer		
#Genes	SVM	kNN	#Genes	SVM	kNN	#Genes	SVM	kNN
30	90.29	92.23	6	95.24	95.24	4	90.20	96.08
	93.0	92.0		55.17	65.52		90.20	94.12
44	95.15	91.26	10	95.24	90.48	5	86.27	96.08
	92.0	92.0		58.62	62.07		90.20	96.08
107	94.17	91.26	20	100.0	100.0	10	92.16	96.08
	93.0	92.0		48.28	58.62		94.12	92.16
160	96.12	91.26	40	85.71	100.0	55	96.08	94.12
	97.0	93.0		62.07	62.07		94.12	92.16

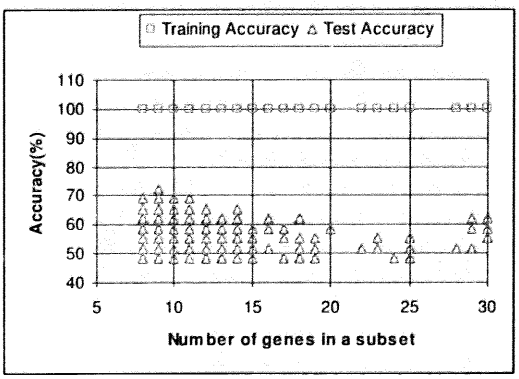


(a) Gene subsets selected by RPMBGA

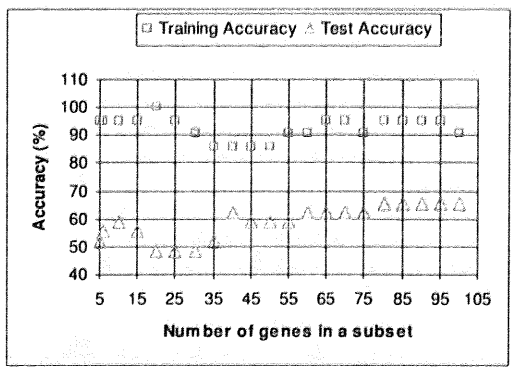


(b) Gene subsets selected by SNR

Figure 5.1: Plot of training and test accuracies of different gene subsets of lung carcinoma data

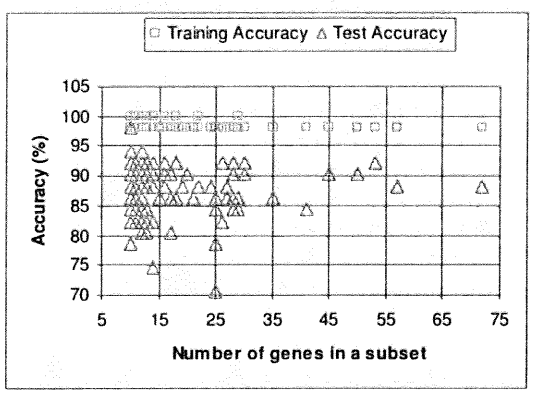


(a) Gene subsets selected by RPMBGA

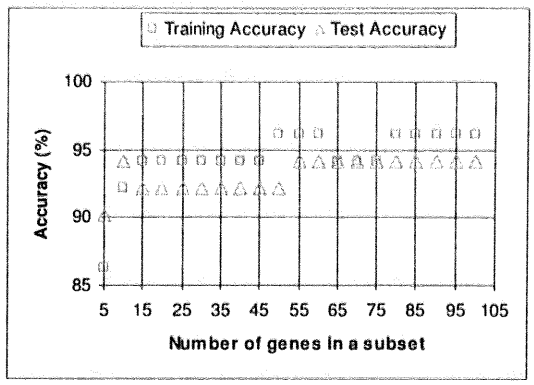


(b) Gene subsets selected by SNR

Figure 5.2: Plot of training and test accuracies of different gene subsets of brain cancer data



(a) Gene subsets selected by RPMBGA



(b) Gene subsets selected by SNR

Figure 5.3: Plot of training and test accuracies of different gene subsets of prostate cancer data

Table 5.10: Description of the selected genes in the best subset of brain cancer data

Feature#	Accession#	Description of Gene
1318_at	X74262	RBBP4: retinoblastoma binding protein 4
32334_f_at	AB009010	Homo sapiens mRNA for polyubiquitin UBC
36699_at	AF023450	human CHD2-52 down syndrome cell adhesion molecule
36847_r_at	AA121509	AA121509:zk88c10.s1 Homo sapiens cDNA, 3 end
38314_at	AB002304	capicua homolog (Drosophila)
40132_g_at	D89937	Homo sapiens mRNA for follistatin-related protein (FRP)
40953_at	S80562	calponin 3, acidic
41790_at	AL031230	glycosylphosphatidylinositol specific phospholipase D1
41837_at	AA149431	AA149431:zl26a08.s1 Homo sapiens cDNA, 3 end

Table 5.11: Description of the selected genes in the best subset of prostate cancer data

Feature#	Accession#	Description of Gene
32560_s_at	W30959	zc65h10.r1 Soares_fetal_heart_NbHH19W cDNA clone
32898_at	U20582	actin like protein
34678_at	AL096713	FER1L3: fer-1-like 3, myoferlin
37639_at	X07732	HPN: hepsin (transmembrane protease, serine 1)
37912_at	X80200	TRAF4: TNF receptor-associated factor 4
38203_at	U69883	KCNN1: potassium intermediate/small conductance calcium-activated channel
39043_at	AF006084	ARPC1B: actin related protein 2/3 complex, subunit 1B, 41kDa
39408_at	Z80345	ACADS: acyl-Coenzyme A dehydrogenase, C-2 to C-3 short chain
39545_at	U22398	CDKN1C: cyclin-dependent kinase inhibitor 1C (p57, Kip2)
41504_s_at	AF055376	MAF: v-maf musculoaponeurotic fibrosarcoma oncogene homolog

Overfitting on microarray data

Overfitting (training accuracy$\gg$test accuracy) is a major problem in classification of microarray data. If the training and test samples are not carefully divided, it may happen during classification of microarray data that one gets 100% accuracy on training data but 0% accuracy on test data. If we look at Table 5.8, and figures 5.1-5.3, we find that overfitting has occurred in both SVM and kNN accuracies. The big differences between training and test accuracies can be observed on brain cancer data. The poor performance of classification models on this data set may be due two reasons: first, the number of training samples is smaller than the number of test samples; second, the expression values of test samples may be totally different from those of training samples since test samples are of non-classic gliomas. However, the classifier, not the gene selection algorithm RMPBGA, is responsible for overfitting because RPMBGA proceeds depending on the accuracy returned by the classifier; if a classifier assigns higher accuracy to an irrelevant gene subset, RPMBGA will assign better fitness to it and eventually select it as a parent for reproduction of new offspring in next generation, which in turn may

produce irrelevant gene subsets causing overfitting. It is the characteristic of a good classifier that assigns higher accuracy to relevant gene subsets but lower accuracy to irrelevant gene subsets.

5.5 Summary

In this chapter, we have investigated the performance of two gene selection methods: RPMBGA and SNR with two classifiers: SVM and kNN on three microarray data sets. By performing experiments, we have found that no single gene exists that can classify patient samples 100% accurately, and neither can all genes. However, by applying our gene selection method, we found that some gene subsets having very small number of genes could classify samples more accurately than single genes or all genes or the genes selected by SNR using either of the classifiers. Moreover, we found many combinations of gene subsets that, having or not equal number of genes, produced the same classification accuracy. These findings suggest that there are many redundant or irrelevant genes in microarray data, and some of them act negatively on the acquired accuracy by the relevant genes, and selection of some genes using only a score metric, independently of the classifier, fails to include only the predictive genes.

We observed during our experiments that whatever gene selection method or classifier is used, greater number of genes is required for classification of multitype tumor samples like lung carcinoma. This is desirable because the information useful for multiclass tumor classification is encoded into complex gene expression patterns that cannot be captured by a small number of genes.

From the experimental results, we found that the best accuracy by the gene subsets selected by RPMBGA was superior to the accuracy of the gene subsets selected by SNR. It is very natural that the probability of finding a good gene subset from a collection of many possible solutions is higher than the probability of finding that one from a limited number of solutions. This suggests that evolutionary computation like RPMBGA can be used as an alternative to widely used SNR for the identification of the highly discriminative genes from microarray data.

Chapter 6

Classification and Gene Selection by Genetic Programming

6.1 Introduction

When the objective of a research is to extract possible bio-markers by mining the gene expression data, we can employ the random probabilistic model building genetic algorithm (RPMBGA) with a suitable classifier like support vector machine (SVM) or k-nearest neighbor (kNN) classifier. However, we have found in the previous chapter that the selected genes as well as the classification accuracy are very much dependent on the choice of the classifier. For some data sets, SVM with RPMBGA produces better accuracy while for some other data sets, kNN with RPMBGA may be a good choice. Finding of an optimal ensemble of gene selection algorithms and classifiers is difficult. Instead of two methods, we can consider one method for two tasks— classification of gene expression data and selection of informative genes. In this context, genetic programming (Koza, 1992; Banzhaf *et al.*, 1998), an evolutionary computation method, is a well-suited candidate method. In its typical implementation, a set of classification rules is produced in multiple runs using the training data, and then the best fitted rule(s) is (are) used as the predicted model that is evaluated using the test data. In addition, the set of rules is analyzed to get the more frequently selected genes that are conjectured to be the possible bio-markers of the cancer under study.

The advantages of genetic programming are as follows:

- The transparent algebraic rules of genetic programming provide an insight

into the quantitative relationships among the genes in classification of samples.

- Genetic programming acts as a classifier as well as a gene selection algorithm. During evolution of classification rules, GP automatically selects some genes from a pool of several thousand genes; this is advantageous because in most other classification methods, we need two systems: a gene selection method and a classifier, and the optimal tuning of all the parameters of these two algorithms is sometimes difficult.

6.2 Genetic programming

Genetic programming (Koza, 1992) is an extension of the genetic algorithm (GA) in which the genetic population consists of computer programs. The basic difference between genetic algorithm and genetic programming is the representation of an individual in the genetic population. In GA, an individual is usually a fixed-length string of symbols whereas in GP, an individual is a variable length tree composed of functions and variables. Thereby, the crossover and the mutation operators are applied in different ways. In gene expressions based classification, the individuals in a GP population are S-expressions of classification rules consisting of functions and variables corresponding to the genes of a microarray data set. Let the S-expression of a rule be represented by R_{expr}, and its output on each sample is a real-valued number. In the typical implementation of a binary genetic programming classifier, the class of a sample Y is predicted as follows:

$$Class(Y) = \begin{cases} \text{'A'} & \text{if } R_{expr}(Y) \geq 0; \\ \text{'B'} & \text{if } R_{expr}(Y) < 0. \end{cases} \tag{6.2.1}$$

That is, the rule for the problem is:

$$\text{IF } R_{expr}(Y) \geq 0 \text{ THEN 'A' ELSE 'B'.}$$

An example of R_{expr} is as follows:

$$(2 * X2474 - X1265/X1223)$$

where $X2474$, $X1265$ and $X1223$ correspond to the expression levels of genes at indexes 2474, 1265 and 1223, respectively in a data set. Genetic programming breeds a population of individuals to solve the problem of classification by executing the following steps:

1. Generate initial population of random compositions of functions and terminal sets (genes).
2. Execute each individual in the population on the training samples and assign it a fitness.
3. While termination criteria is not met, do the following sub steps:
   (a) Create new offspring by repeatedly applying the following three operations to the parents that are selected from the population with a probability based on fitness:
      i. Reproduction: copy a selected parent to the new population without any modification.
      ii. Crossover: create two offspring for the new population by genetically recombining randomly chosen parts of the two selected parents.
      iii. Mutation: create an offspring for the new population by randomly mutating a part of the selected parent.
   (b) Execute each offspring in the new population on the training samples and assign it a fitness.

There are many parameters like the population size, maximum depth of a rule, crossover depth, crossover probability (p_c), reproduction probability (the probability that a selected parent will be copied to the new population without any genetic operation) (p_r), mutation probability (p_m), etc. associated with genetic programming. Detailed descriptions on genetic programming can be found in (Koza, 1992; Banzhaf *et al.*, 1998).

6.2.1 Components of an S-expression in GP

Each S-expression in a population consists of randomly chosen functions and genes. For simplicity, each gene in the expression is represented by an 'X' followed by the index number of the gene in the data set. For example, $X1314$ represents the gene at index# 1314 of a data set. As functions, arithmetic and/or logical functions can used for evolution of classification rules. During the coding of genetic programming, we have to choose an appropriate function set depending on the targeted output. If we want Boolean outputs (either TRUE or FALSE), we can consider a set of functions consisting of either arithmetic and logical functions like {*+, -, *, /, sqr, sqrt, exp, and, or, not, >, >=, <, <=, =*} or only Boolean functions like {*and, or, not, xor, >, >=, <, <=, =*}. If our targeted output is real, we consider only arithmetic functions like {*+, -, *, /, sqr, sqrt, ln, exp, power, sin, cos, tan*}.

6.2.2 Generation of initial population

Initial population of individuals (S-expressions) is generated by random valid composition of functions and terminals. For any position in an S-expression, the choice of a function or a terminal is random with the restriction on the size and the semantic of a rule. The pseudocode of creating a rule in'grow mode' is as follows:

```
GenerateRule(Tree t, Integer depth)
  If (depth < 1) Then return;
  ElseIf (depth = 1) Then
    t.value=SelectTerminalRandomly(terminal_set);
    t.left=null; t.right=null;
    return;
  Else
    node=SelectNodeRandomly(terminal_set+function_set);
    If (node is a terminal) Then
      t.value=node;
      t.left=null; t.right=null;
      return;
```

ElseIf (*node* is a unary function) **Then**
t.value=*node*;
t.right=**null**;
t.left=**new** Tree();
GenerateRule(*t.left,depth-1*);
Else
t.value=*node*;
t.right=**new** Tree();
t.left=**new** Tree();
GenerateRule(*t.left,depth-1*);
GenerateRule(*t.right,depth-1*);

An example of generating a tree of S-expression of maximum depth 5 from the function and terminal sets of $\{+, -, *, /, sqr\}$ and $\{X1, X2, X3\}$ is shown in Fig. 6.1. First the multiplication function '*' is randomly chosen. Since '*' is a binary function, it needs two arguments. In the next steps, the terminal 'X1' is chosen as its left argument and '+' as the second argument. This continues until all the functions in the tree get their arguments.

6.2.3 Evaluation of a rule

The success of an evolutionary computation method is very much dependent on the fitness function used to measure the goodness of an individual. For a binary classification problem, the accuracy can be used as the raw fitness measure of a rule and the standardized fitness can be calculated as follows:

$$fitness(rule) = \frac{1}{1 + (\#TS - \#CCTS)} \tag{6.2.2}$$

where $\#TS$ and $\#CCTS$ are respectively the number of training samples, and the number of correctly classified training samples by that rule.

Matthews (1975) proposed correlation between the prediction and the observed reality as the measure of raw fitness of a predicting program. For a binary classification problem, the correlation (C) is defined as follows:

$$C = \frac{N_{tp}N_{tn} - N_{fp}N_{fn}}{\sqrt{(N_{tn} + N_{fn})(N_{tn} + N_{fp})(N_{tp} + N_{fn})(N_{tp} + N_{fp})}} \tag{6.2.3}$$

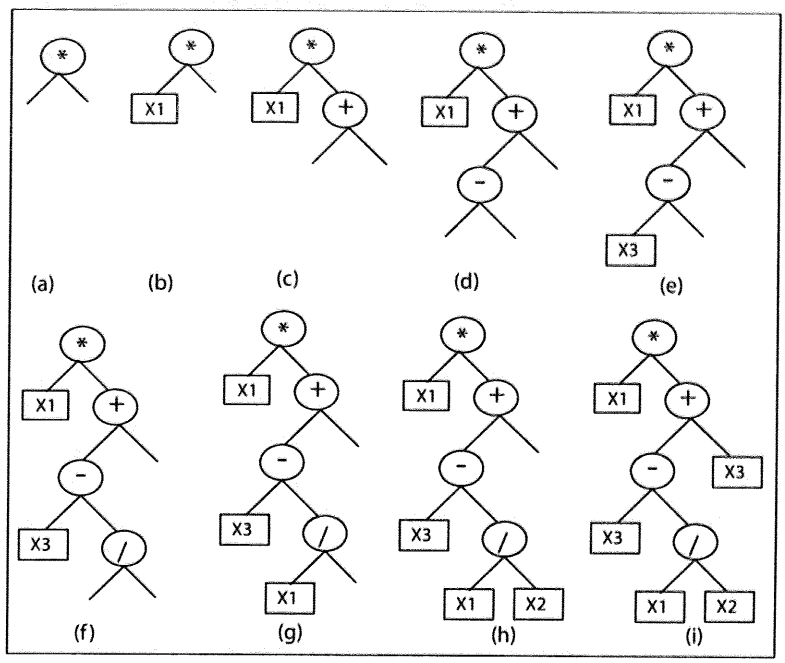


Figure 6.1: An example of creation of an S-expression in GP

where N_{tp}, N_{tn}, N_{fp} and N_{fn} are the number of true positives, true negatives, false positives and false negatives, respectively. When the denominator of equation (6.2.3) is 0, C is set to 0. The standardized fitness of a rule is calculated as follows:

$$fitness(rule) = \frac{1+C}{2}. \tag{6.2.4}$$

Since C ranges between -1.0 and +1.0, the standardized fitness ranges between 0.0 and +1.0, the higher values being the better and 1.0 being the best. In Fig. 6.2, we have shown how the fitness of a genetic programming rule is calculated. The graphical plots of two fitness functions are shown in Fig. 6.3. It appears that fitness calculation using (6.2.4) will be preferable to that by using (6.2.2).

The ultimate objective of a GP is to find a rule that can classify all the samples correctly and thus has fitness=1.0. During execution of the S-expression of a rule on a sample, we take precautions so that the two functions 'sqrt' and '/' do not produce undefined results. In the case of undefined results, we treat them as follows: $\frac{x}{0} = 1$, and $\sqrt{x} = 0$ if $x < 0$. Note that after adjustment, $\sqrt{(x)^2} \neq (\sqrt{x})^2 \neq x$. For example, if $x = -3$, then $\sqrt{(x)^2} = 3$ while $(\sqrt{x})^2 = 0$. Similarly, $z * (x/y) \neq (z * x)/y$; if $y = 0$, then $z * (x/y) = z$ while $(z * x)/y = 1$.

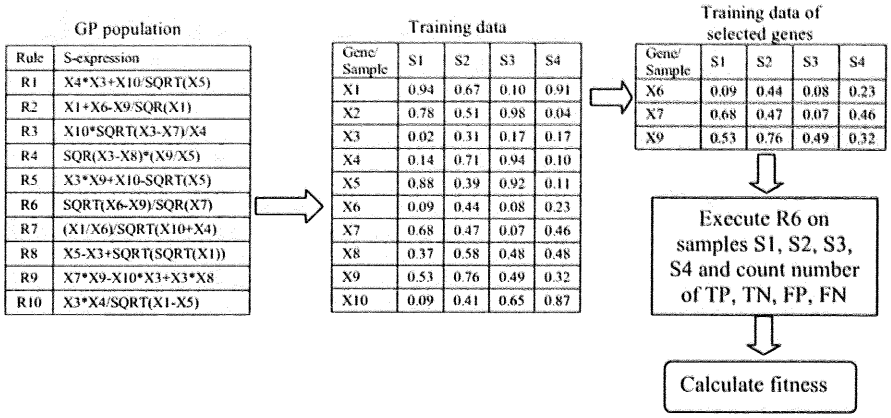


Figure 6.2: Fitness evaluation of a GP rule (R6)

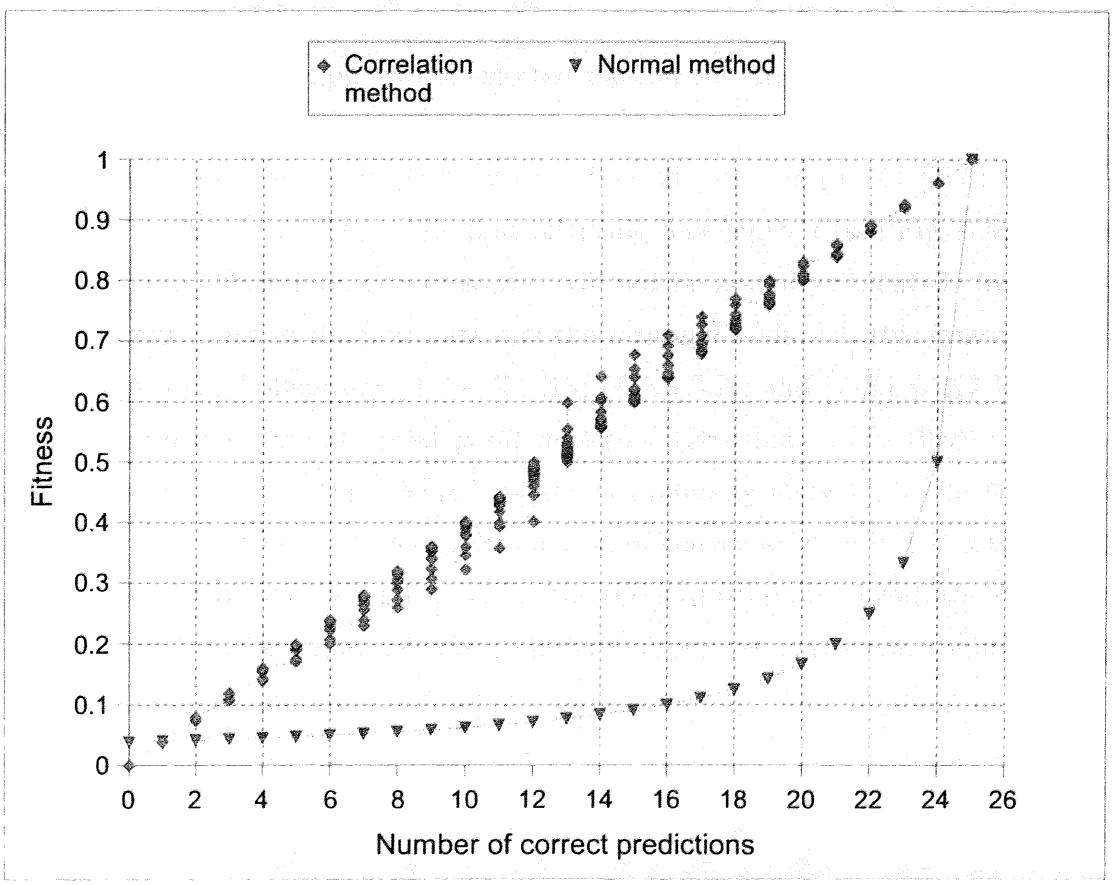


Figure 6.3: Graphical plots of two fitness functions

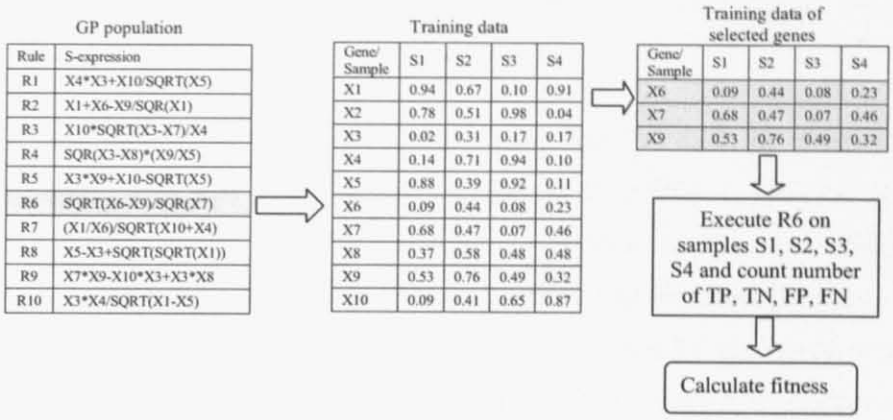


Figure 6.2: Fitness evaluation of a GP rule (R6)

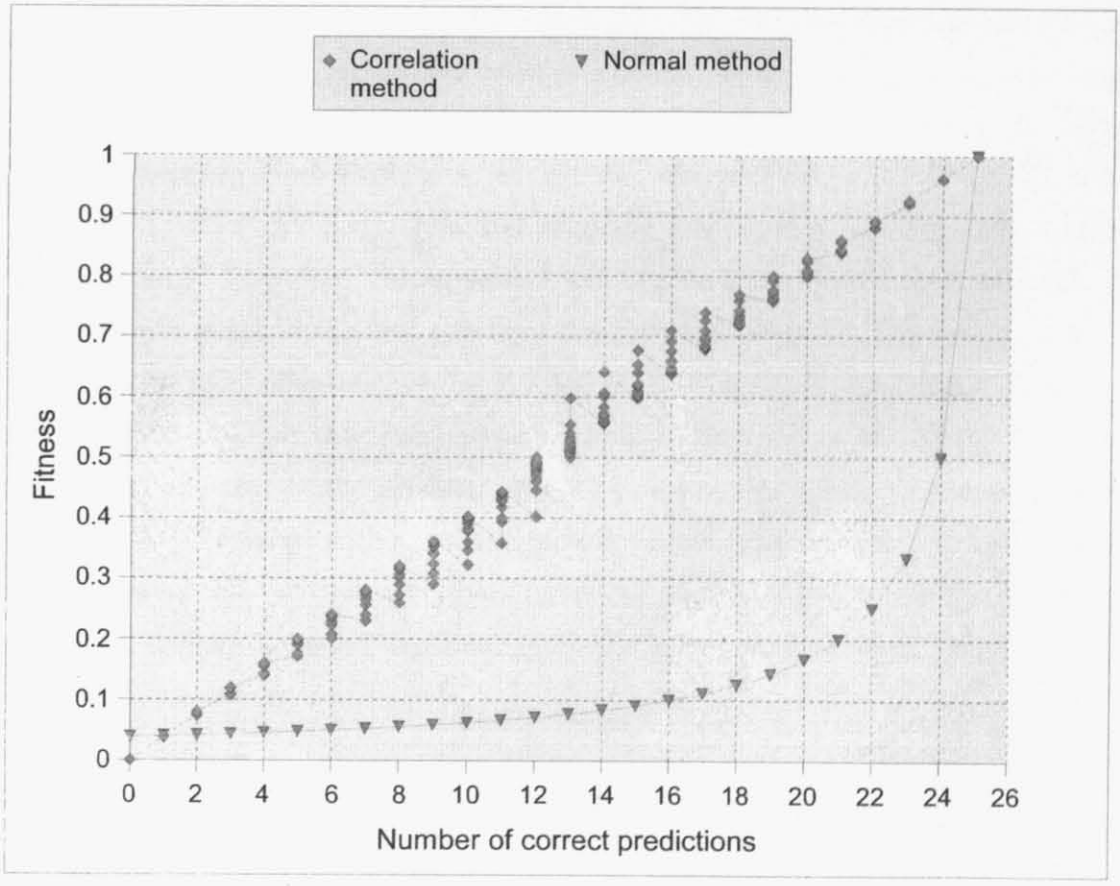


Figure 6.3: Graphical plots of two fitness functions

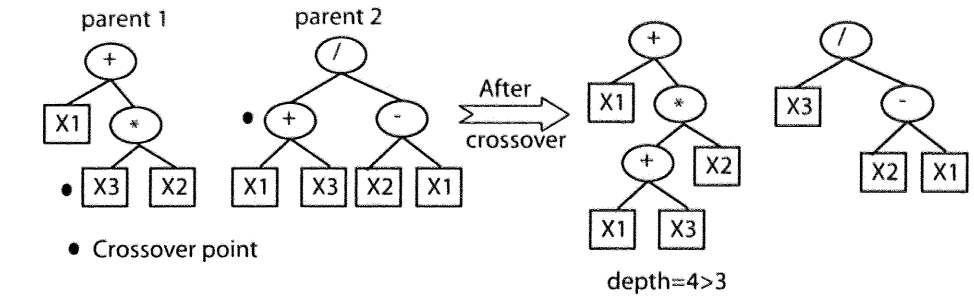
6.2.4 Offspring generation through crossover and mutation

Since the population size is usually very large for the task of classification of gene expression data, the *greedy-over selection method* (Koza, 1992) is applied to select two parents (individuals) for branch-type crossover. Then two offspring are generated by exchanging randomly chosen parts of the selected parents. Usually, a subtree of the first parent is chosen randomly. Then in the second parent, a subtree is randomly chosen with the restriction that when it is exchanged with the chosen subtree of the first parent, their exchanges will generate two valid trees of proper depth. Let us give an example of generating two offspring through crossover. Suppose two trees of two selected parents in preorder traversal format are (+ X1 (* X3 X2)) and (/ (+ X1 X3) (- X2 X1)), and the maximum allowable depth of a tree is 3. Suppose the selected part in the first parent is the terminal node $X3$, and that in the second parent is the subtree containing (+ X1 X3). After exchanging these two parts, we get two offspring as (+ X1 (* (+ X1 X3) X2)) and (/ X3 (- X2 X1)). The first offspring has depth 4 (see Fig. 6.4), which is not allowed. Therefore, this crossover will not be actually executed. Instead, if the crossover point in the first parent is the terminal node $X1$, this crossover will produce two valid offspring: (+ (+ X1 X3) (* X3 X2)) and (/ X1 (- X2 X1)).

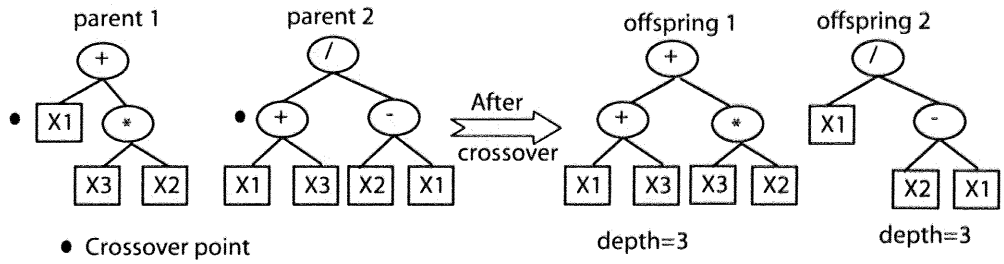
For mutation, we have used *point mutation* (Banzhaf *et al.*, 1998). In it, a node from the tree of the selected parent is randomly chosen. If the node is a function, it is replaced with another function of the same type; if it is a terminal, it is replaced with another terminal. After the mutation operation, the depth of the tree remains the same; however, after the crossover operation, the depths of the trees may change.

6.2.5 Evolution of rules for multiclass classification

By a single rule, we can classify data into two groups. For multiclass data, we need to develop multiple rules for classification. There are two widely used techniques



(a) Example of a crossover that will not be actually executed (depth>max_depth)



(b) Example of a crossover that will be actually executed (all depths<=max_depth)

Figure 6.4: An example of crossover in genetic programming

for classification of multi-type data: one-vs-one and one-vs-rest methods.

In one-vs-one method, if there are c classes in the data, $c(c-1)/2$ rules are developed. Each $rule_{ij}$ is evolved using the training data from two classes (i, j), and the class of a test sample is predicted by 'winner-takes-all' voting strategy. If the rule says that the test sample is in class i, the vote for the ith class is increased by one, else the vote for jth class is increased by one. Then the test sample is predicted to be in the class that has the highest votes.

In one-vs-rest strategy, only c rules are developed by c GP runs—one rule for each class. During evolution of a rule i, the samples of class i are treated as positive; other samples as negative. In this case, the measures: true positive (TP), true negative (TN), false positive (FP) and false negative (FN) for the fitness calculation using equation (6.2.4) of rule i are determined as follows:

IF $(O(Y) \geq 0)$ AND (CLASS(Y)=i) THEN TP;
IF $(O(Y) < 0)$ AND (CLASS$(Y)\neq i$) THEN TN;
IF $(O(Y) \geq 0)$ AND (CLASS$(Y)\neq i$) THEN FP;
IF $(O(Y) < 0)$ AND (CLASS(Y)=i) THEN FN;

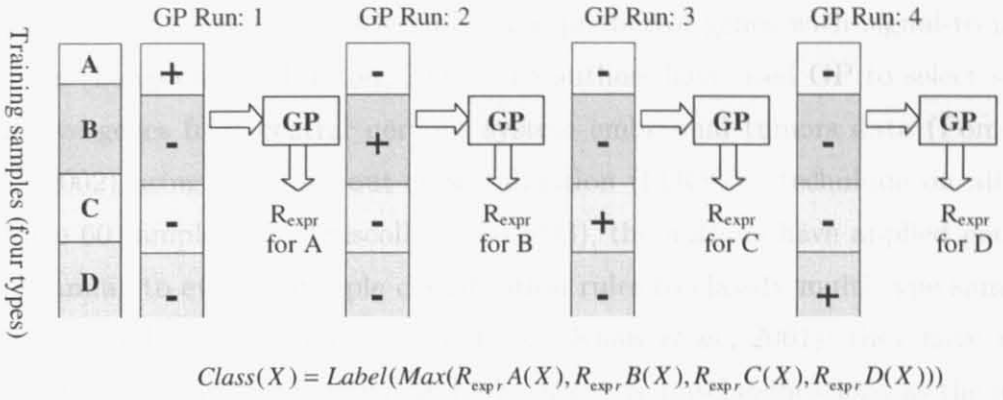


Figure 6.5: One-vs-rest approach for evolution of rules for multi-class classification

where $O(Y)$ is the output of the S-expression of rule i on a test sample Y. Afterwards, the rules are applied to the training and test samples to get generalized accuracy on them. If a GP is trained well, only one rule will fit a sample. If more than one rule fits a sample, the class is predicted depending on the outputs of the rules. If the outputs of the rules are real values, the class of a sample is predicted to be the class that has the highest value of its rule. If two or more rules have the same value or outputs are boolean values, the sample gets the label of the class that has the highest number of samples in the training data. The intuition behind this is that the class having maximum number of training samples should always get priority. If none of the rules fits a sample, the sample is treated as misclassified (an error). An example of one-vs-rest approach for evolution of genetic programming rules for classification of multi-category samples is given in Fig. 6.5.

6.3 Related works with genetic programming

In the past, genetic programming has been used for classification of real and artificial data containing smaller number of attributes (Muni *et al.*, 2004; Tan *et al.*, 2002; Chien *et al.*, 2002; Falco *et al.*, 2002). Recently, it has been applied to analysis of microarray data sets containing huge number of genes (Moore *et al.*, 2002; Hong and Cho, 2004; Driscoll *et al.*, 2003; Langdon and Buxton, 2004). In (Hong and Cho, 2004), the authors have applied GP on the lymphoma data set

(Alizadeh *et al.*, 2000) after selecting some predictive genes with signal-to-noise ratio. In (Langdon and Buxton, 2004), the authors have used GP to select some predictive genes from central nervous system embryonal tumors data (Pomeroy *et al.*, 2002) using leave-one-out-cross-validation (LOOCV) technique on all the available 60 samples. In (Driscoll *et al.*, 2003), the authors have applied genetic programming to evolve multiple classification rules to classify multi-type samples of small round blue-cell tumors (SRBCTs) (Khan *et al.*, 2001); they have used weighted average of false positive and false negative misclassifications as the measure of goodness of a set of rules, the truth table method to combine multiple rules into a single classification rule.

6.4 Evaluation of genetic programming classifier

We applied genetic programming to the classification of four microarray data sets of binary- and multi-type tumors. The data sets include brain cancer (Nutt *et al.*, 2003), prostate cancer (Singh *et al.*, 2002), small round blue-cell tumors (SRBCTs) (Khan *et al.*, 2001), and lung carcinoma (Bhattacharjee *et al.*, 2001) data. The numbers of training and test samples of brain cancer, prostate cancer, SRBCTs, and lung carcinoma are (21, 29), (51, 51), (63,25) and (103,100), respectively. This split of each data set was fixed during the experiments. The settings of different GP parameters are shown in Table 6.1.

To compare the classification accuracy of genetic programming with the classification accuracy of evolutionary and non-evolutionary methods, we applied the kNN (k=11) classifier to the data sets after selecting some discriminative genes with signal-to-noise ratio and with RPMBGA (Chap. 5). For brain and prostate cancers data, we performed 20 experiments with the kNN classifier using $5, 10, 15, \ldots, 100$ genes selected by signal-to-noise ratio (SNR). Similarly, for SRBCTs and lung carcinoma data, we performed 20 experiments with the kNN classifier using $10, 20, 30, \ldots, 200$ genes selected by SNR. We use the notation 'kNN+SNR' and 'kNN+RPMBGA' to denote the application of the kNN classifier on the expression values of the genes selected by SNR and by RPMBGA,

Table 6.1: Typical GP parameter settings

Parameter	Setting
Population size	4000
Maximum number of nodes in a GP tree	100
Function set	{+, -, *, /, SQR, SQRT}
Maximum initial depth	6
Initial population generation method	Ramped half-and-half
Fitness evaluation	Eq. (6.2.2)
Selection method for crossover	Greedy-over (Koza, 1992)
Maximum number of generations per run	100
Maximum crossover depth	7
Reproduction probability	0.1
Crossover probability	0.9
Mutation probability	0.1
Termination criteria	fitness=1.0 or maximum number of generations
Regeneration type	Elitism (elite size=1)

respectively.

For each data set and each type of experiment, we have investigated the biological significance of all the genes selected by the respective method in 20 runs (sets of runs). Some of the genes that are potential biomarkers of the cancers being studied here or more frequently occur in tumorigenesis are described in this chapter. During description of biological significance of a gene, we have used the format 'GeneId (GeneName) [Accession]' where *GeneId* is the feature# or index (if feature# is not available) of the gene in the data set, *GeneName* is the official name of the gene and *Accession* is the GenBank reference for the gene. However, if a gene is referred somewhere before, we have used only the real name of that gene.

6.4.1 Results

We performed experiments on the data divided into training and test samples. During the learning phase of genetic programming, only the training samples were

Table 6.2: Training accuracies on the four microarray data sets

Data Set	Genetic Programming		kNN+RPMBGA		kNN+SNR	
	Best	Average	Best	Average	Best	Average
Brain cancer	100.0	$100 \pm 0^{\ddagger}$	100.0	100.0 ± 0	100.0	91.67 ± 2.62
Prostate cancer	100.0	$95.69 \pm 2.17^{\ddagger}$	98.04	96.08 ± 1.56	92.16	90.69 ± 1.25
SRBCTs	100.0	$100.0 \pm 0^{\dagger\ddagger}$	100.0	99.21 ± 0.81	98.41	97.70 ± 1.21
Lung carcinoma	99.03	$96.26 \pm 2.96^{\dagger\ddagger}$	92.23	90.73 ± 0.86	90.29	86.12 ± 4.39

Table 6.3: Test accuracies on the four microarray data sets

Data Set	Genetic Programming		kNN+RPMBGA		kNN+SNR	
	Best	Average	Best	Average	Best	Average
Brain cancer	79.31	$66.90 \pm 7.20^{\dagger\ddagger}$	75.86	59.14 ± 9.51	48.28	48.45 ± 0.77
Prostate cancer	88.24	82.75 ± 5.03	88.24	$88.63 \pm 1.87^{\dagger}$	90.20	$88.82 \pm 0.92^{\ddagger}$
SRBCTs	100.0	$80.40 \pm 8.20^{\dagger\ddagger}$	80.0	72.0 ± 5.19	76.0	74.40 ± 4.57
Lung carcinoma	87.0	80.95 ± 4.29	92.0	$89.60 \pm 3.03^{\dagger}$	94.0	$92.0 \pm 2.75^{\ddagger}$

used, and the test samples were totally isolated from the training samples. We also performed experiments using the kNN classifier after selection of genes using signal-to-noise ratio and using RPMBGA as described before.

In tables 6.2 and 6.3, the training and test accuracies of different algorithms on four microarray data sets are summarized; in table 6.4, the descriptions of some of the more frequently occurring genes in the best 20 rules (sets of rules) of 20 independent runs (for SRBCTs and lung carcinoma, 80 and 100 runs, respectively) are presented. Since the test set is a blind set during learning of a classifier, we define the best test accuracy as the highest accuracy on the test data corresponding to the highest training accuracy. For example, if two gene subsets or rules both produce 100% accuracy on training data but get 87 and 90% accuracy, respectively, on test data, the best training and test accuracies would be 100 and 90%, respectively. The significantly higher average accuracy (at 5% level of t-test) of GP and kNN+RPMBGA is indicated by a superscript '†' while that of GP and kNN+SNR is indicated by a superscript '‡' in the tables.

In terms of training and test accuracies, GP is superior to the other two methods on brain cancer and SRBCTs data. On prostate cancer data, though GP obtains significantly higher training accuracy than kNN+SNR, it obtains

significantly lower test accuracy than the other two methods (kNN+SNR and kNN+RPMBGA). On prostate cancer data, GP only gets better training accuracy than kNN+SNR but it gets significantly lower test accuracy than other two methods. On lung carcinoma data, GP obtains significantly higher training accuracy than other two methods but it gets significantly lower test accuracy than other two methods. On lung carcinoma data, kNN+SNR underfits (training accuracy<test accuracy) the training data. The summary of the different evolved best rules by GP and the best gene subset selected by SNR or RPMBGA is given below.

For brain cancer data, we got 20 rules that are able to classify all the 21 training samples correctly but they fail to classify the 29 test samples 100% accurately. The best rule found by GP is as follows:

- IF $(((245\_at)^2 - \sqrt{32809\_at} * (38699\_at - (35915\_at - 37308\_at)^2)) * ((245\_at)^2 - (32275\_at)^2 - \sqrt{38398\_at + 35275\_at})) \geq 0$ THEN 'AOD' ELSE 'GB'.

This rule can classify all the samples except the six test samples: 35, 40, 41, 42, 46 and 47. The two genes 32275_at (SLPI) [NM_003064] and 38398_at (MADD) [NM_003682] of this best rule have roles in cancer diseases. SLPI regulates cancer development (Devoogdt *et al.*, 2004) while MADD suppresses tumor cell survival and enhances susceptibility to apoptosis and cancer therapy (Efimova *et al.*, 2004). The more frequently occurring genes in 20 rules are given in Table 6.4. Among these genes, 226_at (PRKAR1A)[NM_002734] is a tumor-suppressor gene for sporadic thyroid cancer (Sandrini *et al.*, 2002), and 41753_at (ACTN4) [NM_004924] promotes tumorigenicity and regulates cell motility of human lung carcinoma (Menez *et al.*, 2004). However, the relationships of these genes with brain cancer are yet unknown. Using kNN with signal-to-noise ratio, we got the best 20-gene subset that can classify training and test samples, respectively, 100 and 48.28% accurately. The kNN classifier with RPMBGA got an 8-gene subset that produces the best 100 and 75.86% training and test accuracies, respectively. Some of the more frequently occurring genes in the 20 best gene-subsets selected by RPMBGA with the kNN classifier in 20 runs are 33619_at (RPS13) [NM_001017], 32272_at (K-ALPHA-1) [NM_006082], 34091_s_at

Table 6.4: Descriptions of the genes more frequently selected during training and test accuracy estimation on the microarray data sets

Data set	Accession#	Gene description	Freq
Brain cancer	NM_015853	LOC51035: unknown protein LOC51035	3
	NM_002734	PRKAR1A: Human cAMP-dependent protein kinase type I-alpha subunit	2
	NM_006703	NUDT3: nudix (nucleoside diphosphate linked moiety X)-type motif 3	2
	NM_004924	ACTN4: actinin, alpha 4	2
	NM_002086	GRB2: growth factor receptor-bound protein 2	2
Prostate cancer	NM_006159	NELL2: NEL-like 2 (chicken)	9
	NM_003039	SLC2A5: solute carrier family 2(facilitated glucose/fructose transporter), member 5	8
	NM_002899	RBP1: retinol binding protein 1, cellular	7
	NM_002825	PTN: pleiotrophin (heparin binding growth factor 8, neurite growth-promoting factor 1)	6
	NM_015101	GLT25D2: glycosyltransferase 25 domain containing 2	6
	NM_000894	LHB: luteinizing hormone beta polypeptide	5
	NM_014350	TNFAIP8: tumor necrosis factor, alpha-induced protein 8	5
SRBCTs	NM_001792	CDH2: cadherin 2, type 1, N-cadherin (neuronal)	11
		ESTs	10
	NM_002011	FGFR4: fibroblast growth factor receptor 4	8
	NM_006765	TUSC3:tumor suppressor candidate 3	8
	NM_002402	MEST: mesoderm specific transcript homolog (mouse)	7
Lung carcinoma	NM_003463	PTP4A1: protein tyrosine phosphatase type IVA, member 1	25
	NM_001723	DST: dystonin	14
	NM_003665	FCN3: ficolin (collagen/fibrinogen domain containing) 3 (Hakata antigen)	13
	NM_001024847	TGFBR2: transforming growth factor, beta receptor II (70/80kDa)	11
	NM_004787	SLIT2: slit homolog 2 (Drosophila)	11
	NM_003617	RGS5: regulator of G-protein signalling 5	10
	NM_003278	CLEC3B: C-type lectin domain family 3, member B	10
	NM_001336	CTSZ: cathepsin Z	8

(VIM) [NM_003380] and 327_f_at (RPS20) [NM_001023]. The genes RPS13, VIM and RPS20 are also included in the best gene-subset found by SNR with kNN. However, none of these genes are known to be involved in brain cancer.

For prostate cancer data, we got only two rules that can classify all the 51 training samples accurately; however, they are not able to classify test samples 100% correctly. The best rule that can classify 45 test samples accurately is as follows:

- IF $(1662\_r\_at/((4*32242\_at)/(37639\_at - 34811\_at + 32252\_at) + 32242\_at) - 2075\_s\_at + 37639\_at - 35642\_at - 34811\_at + 32252\_at - 2*32242\_at) \geq 0$ THEN 'PT' ELSE 'NL'.

This rule fails to correctly classify 6 prostate tumor test samples: 64, 68, 82, 84, 90, 95. Interestingly, this rule can classify all the normal samples correctly. Among the genes in this best rule, the gene 37639_at (HPN) [NM_002151] is functionally linked to hepatocyte growth factor/MET pathway and thus may contribute to prostate cancer progression (Kirchhofer *et al.*, 2005). Among the most frequently occurring genes in the best 20 rules of 20 runs, 34820_at (PTN) [NM_002825] acts as an important regulator of diverse biological activities in human prostate cancer cells (Hatziapostolou *et al.*, 2005), 33243_at (TNFAIP8) [NM_014350] is an oncogenic factor in cancer cells (Kumar *et al.*, 2004), and 697_f_at (LHB) [NM_000894] is a weak risk factor for prostate cancer (Elkins *et al.*, 2003). Using kNN with signal-to-noise ratio, we got the best 15-gene subset that can classify training and test samples, respectively, 92.16 and 90.20% accurately. The kNN classifier with RPMBGA obtained an 8-gene subset that produces the best 98.04 and 88.28% training and test accuracies, respectively. Some of the more frequently occurring genes in the 20 best gene-subsets selected by RPMBGA with the kNN classifier in 20 runs are 38406_f_at (ao89h09.x1 Homo sapiens cDNA) [AI207842], 36638_at (CTGF) [NM_001901], 769_s_at (ANXA2) [NM_001002857], 216_at (PTGDS) [NM_000954] and 31444_s_at (ANXA2P3) [NR_001446]. Out of these genes, only ANXA2 has a mechanistic and regulatory role in prostate cancer progression (Banerjee *et al.*, 2003). However, this gene is not included in the best gene-subset

selected by SNR with the kNN classifier.

SRBCTs data set contains four types of samples. In (Khan *et al.*, 2001), the authors used principal component analysis (PCA) (Jolliffe, 2002) and neural networks for classification of the four types of samples and got 100% accuracy on training and test data using a subset of 10 genes. Using genetic programming, we also got a set of four rules that can classify all the samples 100% accurately. The best set of four rules is as follows:

- IF $(X1319 - X2050 - (X1640)^2) \geq 0$ THEN 'EWS'.
- IF $((X123)^2 - \sqrt{(X269)}) \geq 0$ THEN 'BL'.
- IF $((X1311 + X416) * (X2136 + X651) - (X262 + X842)^2) \geq 0$ THEN 'NB'.
- IF $(X1955 - X131 + X129 - (\sqrt{X339}/X1955)) \geq 0$ THEN 'RMS'.

Two genes X262 (BCL7B) [NM_001707] and X1955 (FGFR4) [NM_002011] of these rules are associated with cancer. BCL7B encodes the BCL7A protein that is known to be directly involved in a three-way gene translocation in a Burkitt lymphoma (BL) cell line (Entrez Gene, 2006). FGFR4 is one of the most frequently selected genes by different methods and is overexpressed in gynecological tumor samples, suggesting a role in breast and ovarian tumorigenesis (Jaakkola *et al.*, 1993). Due to its moderate to strong cytoplasmic immunostaining in all RMS samples, it may be a possible bio-marker for RMS (Khan *et al.*, 2001). Among the other more frequently occurring genes, X715 (TUSC3) [NM_006765] is a tumor suppressor gene and it is expressed in most nonlymphoid human tissues including prostate, lung, liver, and colon (Entrez Gene, 2006); X1911 (MEST) [NM_002402] encodes a member of the alpha/beta hydrolase fold family, and the loss of imprinting of this gene is related to tumorigenesis and malignant transformation (Nakanishi *et al.*, 2004). Using kNN with signal-to-noise ratio, we got the best 30-gene subset that can classify training and test samples, respectively, 98.41 and 76% accurately. The kNN classifier with RPMBGA got a 71-gene subset that produces the best 100 and 80% training and test accuracies, respectively. The best gene subsets selected by these two methods also contain the gene FGFR4. Some

of the other more frequently occurring genes in 20 best gene-subsets selected by RPMBGA with the kNN classifier in 20 runs are X545 (CD99) [NM_002414], X246 (CAV1) [NM_001753], X509 (IGF2) [NM_000612], and X153 (RCV1) [NM_002903]. Of these genes, CAV1 and RCV1 are of biological interest because CAV1 inhibits breast cancer growth and metastasis (Sloan *et al.*, 2004) while RCV1 may be the antigen responsible for cancer-associated retinopathy (Ohguro and Nakazawa, 2002).

The lung carcinoma data contains five category of samples. For this data set, we did not find any set of rules that can classify all the training and test samples 100% accurately. The best set of five rules can classify 102 training and 87 test samples accurately (the training and test accuracies are 99.03 and 87%, respectively). We refrain from providing the set of rules here because the rules for 'AD' and 'SQ' are very complex to understand the quantitative relationships among the genes. Among the more frequently occurring genes in the best 20 sets of five rules, 843_at (PTP4A1) [NM_003463], 39634_at (SLIT2) [NM_004787], and 32514_s_at (CTSZ) [NM_001336] are related with cancer. PTP4A1 has a role in tumorigenesis because its overexpression in mammalian cells confers a transformed phenotype (Entrez Gene, 2006); SLIT2 has tumor suppressor activity and is frequently inactivated in lung and breast cancers (Dallol *et al.*, 2002), and CTSZ is expressed ubiquitously in cancer cell lines and primary tumors, and like other members of this family may be involved in tumorigenesis (Entrez Gene, 2006). Using kNN with signal-to-noise ratio, we got the best 130-gene subset that can classify training and test samples, respectively, 90.29 and 94% accurately. The kNN classifier with RPMBGA got a 9-gene subset that produces the best 92.23 and 92% training and test accuracies, respectively. Some of the more frequently occurring genes selected by kNN with RPMBGA in the best 20 subsets are 613_at (KRT5) [NM_000424], 36105_at (CEACAM6) [NM_002483], 700_s_at (MUC1) [NM_001018016], 31950_at (PABPC1) [NM_002568], and 770_at (GPX3) [NM_002084]. Out of these genes, MUC1 allele is associated with susceptibility to lung adenocarcinoma and poor prognosis (Mitsuta *et al.*, 2005).

6.5 Summary

In this chapter, we investigated the applicability of genetic programming for classification of binary- and multi-type tumor data. We performed different experiments with GP, and with the widely used kNN classifier after selection of some genes with signal-to-noise ratio and RPMBGA. We found that GP can evolve simple arithmetic classification rules of gene expressions that can classify patients' samples with encouraging accuracy. In two cases, the results of GP are better than those of other two methods. However, the potential challenge for genetic programming is that it has to search two large spaces of functions and genes simultaneously to find an optimal solution. Therefore, the proper choice of function set, genetic operators, and the depth of a rule is very important in applying GP to classification of microarray data because increasing the depth of a rule and the number of functions may increase the complexity of the rules with little or no improvement in classification accuracy.