Chapter 2

Statistical & Optimization Techniques

QSAR studies can be broadly divided into two types - regression and classification. The development of QSAR models essentially consists of the application of statistical methods to chemical datasets. As such, the statistical and machine learning literature provides a number of useful techniques. Some techniques are specifically designed to build classification models whereas others can carry out both classification as well as regression. In addition to these techniques, a number of methods are available for the optimization of various parameters and selection of variables required in the model building process. These can be deterministic methods such as the BFGS algorithm[1–4] and the Nelder-Mead simplex algorithm[5] or stochastic methods such as genetic algorithms[6–8] and simulated annealing.[9] This chapter discusses the underlying details of the various modeling and optimization techniques used in this work.

2.1 Linear Methods

As the title of this section indicates linear methods employ a linear relationship between the predictor variables and the observed response to develop a predictive model. In many QSAR problems, structure property trends can be modeled reasonably well by linear approximations. In general it is observed that physical properties are well modeled by these types of methods. In the case of biological properties linear models do not always exhibit good predictive performance. The poorer behavior of linear models when faced with biological structure property trends is understandable when we consider the fact that biological properties in general are the result of a number of interactions that might include absorption, metabolic degradation, excretion and so on. Clearly the relationship between molecular structure and these factors is complex and in general nonlinear. However, linear methods are useful as a first step in the modeling process and, though not always very accurate, the simple interpretation methods that can be applied to linear models makes up, to some extent, for the lack of predictive ability for these methods. Though linear methods can be applied to both classification and regression we focus on the latter application in this section.

2.1.1 Multiple Linear Regression

A linear relationship between an observation's response (i.e., observed value) and its independent variables can be modeled by

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i \qquad i = 1, 2, \ldots, n \tag{2.1}$$

where y_i is the response for the i^{th} observation and $x_{i1}, x_{i2}, \ldots, x_{ip}$ are the independent variables for the i^{th} observation and n is the number of observations. $\beta_0, \beta_1, \ldots, \beta_p$ are parameters that are to be estimated. ϵ_i is the error term and is assumed to be a normally distributed random variable. Multiple linear regression is a technique by which y_i and $\beta_0 \ldots \beta_p$ can be estimated. Thus Eq. 2.1 may be written as

$$\hat{y}_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + \ldots + b_p x_{ip} \tag{2.2}$$

where $b_0, b_1, \ldots, b_p$ are the estimated values of the parameters in Eq. 2.1. The most popular algorithm to estimate the parameters is the least squares method which considers the best fitting straight line to be that which minimizes the square of the error between the predicted response, $\hat{y}_i$ and the observed response, y_i.

Once the parameters have been estimated, the model quality can be ascertained in a number of ways. Two common measures of model quality are the R^2 value and the root mean square error (RMSE). The R^2 is also known as the Pearson coefficient and ranges from -1 to +1. The RMSE is defined as

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum (y_i - \hat{y}_i)^2} \tag{2.3}$$

Good models are characterized by high value of R^2 and low values of RMSE. However, it is well known that R^2 is not always a good indicator of model quality and in many cases can be misleading. An alternative to R^2 is Q^2 which is obtained by using a leave-one-out (LOO) cross-validation procedure. That is, the linear model is generated using the whole dataset excluding one point. The response for this point is then predicted using the model and this procedure is repeated for all the points in the dataset. The R^2 for these predictions is denoted by Q^2. Though this is more reliable than R^2, especially for small datasets, care should be taken as it has been shown to be a poor indicator of predictive ability in 3D QSAR[10] and 2D QSAR[11] models. Other indicators of model quality include the F-statistic and partial F-statistics.[12]

Individual predictions can be examined by a variety of methods. The simplest method is to plot residuals (the difference between the observed property and the predicted value) versus the observed response or index number. In either case a good model is characterized by a normal random distribution of residuals. Distinct patterns (such as upward or downward trends) are indicative of heteroscedasity and the dataset must be reexamined. The residuals may also be examined by making a normal probability plot (Q-Q plot). If the residuals do have a normal random distribution, this plot will be a straight line. Deviations from this will indicate shifts in location or scale as well as the presence of outliers.[13] When studentized residuals[12] are used, residuals lying above 2.0 or below -2.0 are traditionally designated as outliers. Outliers can also be determined by the use of regression diagnostics such as the Cooks distance and Mahalanobis distance. Once outliers have been detected the model can be regenerated excluding the outlying compounds from the dataset.

2.1.2 Robust Regression

An alternative to the multiple linear regression algorithm is to use a robust regression algorithm. As mentioned above the least squares algorithm attempts to minimize the squared deviations of the predicted response. This method is characterized by a breakdown point (a measure of the capability of an estimator to tolerate noisy data) of 0%. Robust regression utilizes alternative algorithms characterized by much higher breakdown points. Examples include the least median squares and least trimmed squares algorithms.[14] The advantage in using a robust regression method is that it uses algorithms that dampen the influence of *bad* points and attempts to take into account the whole dataset. Bad (or influential) points will be characterized by high residuals and thus robust regression combines model building and outlier detection in one operation. Thus the three-step process of model building, outlier detection, model regeneration is avoided.

2.2 Nonlinear Methods

Nonlinear methods can be considered a generalization of linear methods. Nonlinear estimation methods do not make any assumptions about the nature of the relationship between the predictor variables and the response. In general, the relationship must be specified in parametric form by the user. When considering nonlinear models, distinction must be made between intrinsically linear and intrinsically nonlinear models. The former class of models can be transformed to a linear form and subsequently analysed using linear methods. Examples of these types of methods include logit and probit regression models. In the latter case, the nonlinear form of the model cannot be transformed to a linear form. Examples of this type of model include the general growth model and models used to determine drug responsiveness and half maximal response. In general, nonlinear models are essentially optimization problems. That is, the parameters are optimized to minimize certain criteria. Some approaches to nonlinear regression include least squares, maximum likelihood and function minimization (quasi-Newton and simplex methods).

In this work we focus specifically on the use of neural network algorithms for nonlinear classification and regression. Neural network algorithms are a specific class of nonlinear methods. They differ from traditional nonlinear methods in the representation of information extracted from the dataset. In contrast to nonlinear methods described above, neural networks do not represent the relationships within the data in an explicit functional form. The relationships in the dataset are encoded by a set of connections between units termed neurons. Methods have been described that attempt to represent this encoding in analytical form,[15] but this is not generalizable to all types of neural network algorithms. Essentially neural network algorithms attempt to mimic the behavior of a human brain and thus an essential feature of these algorithms is the ability to *learn* the relationships present within a dataset. In the words of Haykin[16]

> A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:
>
> 1. Knowledge is acquired by the network from its environment through a learning process.
> 2. Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge.

However, as pointed out by Ripley,[17] the above definition excludes a number of neural network algorithms such as the Kohonen network.[18] The neural network literature describes a large variety of neural network algorithms and Haykin[16] provides an extensive discussion of a variety of neural networks. In this section we focus on the two types of neural networks used in this work, viz., feed-forward neural networks and the self organizing map.

2.2.1 Feed-Forward Neural Networks

The fundamental components of a neural network are neurons. Neurons are essentially computation units that accept an input value and generate an output value. In this sense, a neuron can simply be considered a mathematical function. The actual behavior of the network emerges from the connections between the neurons and weights assigned to these connections.

The neural network models considered in this work are termed 3-layer, fully-connected, feed-forward neural network models (which is a specific case of the multilayer perceptron[16]). The first layer is termed the input layer and each neuron in this layer corresponds to the input variables for the model. The second layer is termed the hidden layer and is responsible for nonlinearly combining the inputs. The final layer is termed the output layer, and in all the neural network models in this work, contains a single neuron whose output is the predicted property. The term fully-connected indicates that all the neurons in a given layer are connected to all the neurons in the next layer. Fig. 2.1 shows a schematic of this type of neural network. Let us now consider the internals of the network in a little more detail.

The role of a neuron is to accept input values and weights associated with connections to the neurons in the preceding layer. Essentially, a neuron consists of a function whose output represents a hyperplane that divides the input space of the neuron into two regions - an *on* region and an *off* region. The function used in a neuron is termed the transfer function and when this function is linear this interpretation holds exactly. However, the main reason for the utility of a neural network is that it exhibits the *universal function approximation* property[19,20] and to achieve this, the transfer function is generally nonlinear in nature. In this case, the above interpretation still holds to a good degree. A number of transfer functions have been reported in the neural network literature and the implementation used in this work utilizes a sigmoidal transfer function given by

$$O = \frac{1}{1 + \exp(-\sum x_i w_i + b)} \tag{2.4}$$

where O is the output of the neuron, x_i is the output value of the i^{th} neuron in the preceding layer, w_i is the weight for the connection between this neuron and the i^{th} neuron in the preceding layer and b is the value of the bias term. Fig. 2.2 is a graphical representation of a hidden neuron. The diagram stresses the fact that the bias term can be considered as a neuron whose output value is always 1. The value of the bias term thus represents the weight for the connection between the *bias neuron* and the hidden neuron in question.

Fig. 2.3 shows a plot of the transfer function defined by Eq. 2.4. The weights provide a means for the neural network to assign importance to specific neurons. In the case of a 3-layer network, weights between the input and hidden layer neurons allow the network to be configured so that more important input variables will have greater contributions to the hidden layer neurons. It should be noted that in this type of configuration, the input layer neurons do not utilize the transfer function. Instead, the job of the input neuron is simply to scale the raw descriptor values to a suitable range (0.05 to 0.95 in the case of the current implementation). The role of the bias term can be understood in terms of a hyperplane interpretation of the transfer function. In this interpretation the bias term plays the role of the intercept term. This view is exact when the transfer function is linear. In effect the bias term shifts the hyperplane in the input space of a neuron to obtain an optimal partitioning of the space into on and off regions. An alternative view is that the bias term controls whether the neuron output is on (0 or close to 0) or off (1 or close to 1). In the case of a neuron with a sigmoidal transfer function, the bias term thus controls where on the sigmoidal function the output will lie. When the output value is close to zero, the neuron will have little contribution to the next layer, whereas when the output value is high, the neuron will have a significant contribution to the next layer. In effect, the weights and biases allow the network to *learn* the features present in the dataset and the optimal set of weights and biases encode these features allowing the network to make accurate predictions.

At this point we have described the anatomy of a neural network. The next step is to obtain a set of weights and biases that will allow the network to make accurate predictions. The number of weights and biases is defined by the configuration of the neural network. In the case of the 3-layer network, the number of weights and biases is defined by the number of input and hidden layer neurons. Intuitively, increasing the number of hidden layer neurons will lead to a more accurate network. However this will also lead to overfitting. One rule of thumb used to determine the suitable number of weights and biases is that the total number of parameters should be less than half the size of the training set used to build the model,[21] that is,

$$n_I n_H + 2n_H + n_O \leq \frac{n_{TSET}}{2} \tag{2.5}$$

where n_I, n_H and n_O are the number of input neurons, hidden neurons and output neurons respectively and n_{TSET} is the number of observations in the training set. The number of neurons in each layer define the *architecture* of the neural network. Thus, using the above notation, a 3-layer neural network is said to have a $n_I - n_H - n_O$ architecture. For all the neural network models reported in subsequent chapters the value of n_O is set to 1.

Once the number of parameters have been chosen, the next step is to obtain a set of optimal parameters. The network is initialized with a set of weights and biases generated using a combination of generalized simulated annealing and the BFGS algorithm. Then, each member of the training set is presented to the network. For each member, the network generates a prediction of the activity or property in question. After each training sample is presented to the network the prediction error is used to update the weights and biases. Traditionally the training procedure utilizes the backpropagation algorithm.[16] However, this algorithm is relatively inefficient and hence we use the BFGS quasi-Newton algorithm.[22] The important feature of the training phase is that it is supervised. That is, to train the network, the observed values of the training samples are required. This is in contrast to unsupervised methods (such as the self-organizing map) that do not require the observed values of the training set. Once all the examples in the training set have been presented to the network the process is repeated for a user specified number of cycles. It is also important to note that since the network is trained using a quasi-Newton algorithm, the optimized weights and biases depend on the initial configuration. As a result multiple runs are required to ensure that a representative result is obtained.

As mentioned above, too many parameters can lead to overfitting. In addition, as training progresses, the training RMSE will continually decrease and after a certain point the network will start to memorize the noise in the dataset. That is, the network will overfit the data. To prevent this, cross-validation is used. This procedure uses a portion of the dataset to measure the performance of the network, in terms of RMSE, at regular intervals during training. In effect, the cross-validation set acts as a pseudo external prediction set. Ideally the RMSE for the cross-validation set will smoothly decrease as training progresses and at one point will start to increase. This point represents the optimal configuration of the weights and biases and any further training beyond this point will lead to overfitting. In practice, the cross-validation RMSE does not always smoothly decrease, but in general, a global minimum does occur. The behavior of the RMSE values for the training and cross-validation sets are shown in Fig. 2.4. The use of the cross-validation set allows us to define a cost function which can be used to assess the quality of a model by simultaneously taking into account its training and cross-validation performance. The cost function in the CNN algorithm used in this work is defined as

$$\text{Cost} = \text{RMSE}_{TSET} + 0.5 \times |\text{RMSE}_{TSET} - \text{RMSE}_{CVSET}| \tag{2.6}$$

where RMSE_{TSET} and RMSE_{CVSET} represent the RMSE values for the training and cross-validation sets. The form of the cost function penalizes models that have overfit, represented by high RMSE for the cross-validation set, and thus characterizes a given model better than simply considering the RMSE of the training set.

Once a model has been trained, its generalizability is then evaluated by passing an external prediction set and noting the RMS error. Ideally, one would expect similar RMSE values for the training, cross-validation and external prediction sets, but in general this is not the case. This aspect of model assessment is discussed in more detail in subsequent chapters.

2.2.2 Kohonen Self-Organizing Maps

A Kohonen self-organizing map (SOM) is an unsupervised neural network that uses only the independent variables of the dataset and is generally applied to classification problems. The SOM was first described by Kohonen[18] in the 1980's. The use of SOM's is widespread and examples of their application include the analysis and prediction of NMR spectra,[23, 24] classification of reactions[25] and QSAR analysis.[26–28]

The SOM can be viewed as an elastic net of points in 2-D, which are molded to the specific features of the compounds used for training. In this sense, the SOM is also a dimension reduction algorithm. Training occurs as the SOM's neurons compete with each other for selection. At each training iteration, the selected neuron and its neighbors are modified to resemble the applied example compound.

SOM's can appear in a variety of forms[18] ranging from a square (or rectangular) grid to a hexagonal array. In this work we use a square configuration. In order that each neuron has the same number of neighbors, the grid is designed so that it wraps around the edges, effectively transforming the grid of neurons into a torus. However, for ease of visualization and discussion we will refer to the arrangement as a square grid.

Each compound in the training set is represented by a vector,

$$X_i = (x_{i1}, x_{i2}, \cdots, x_{in}) \tag{2.7}$$

where n is the number of independent variables employed. Each neuron on the square SOM grid is also a vector,

$$M_i = (m_{i1}, m_{i2}, \cdots, m_{in}) \tag{2.8}$$

where n is defined above. The neurons on the grid are initialized with random vectors. The size of the grid is chosen by trial and error, guided by a rule of thumb described by Chen,[29] which states that the number of neurons should be approximately one to three times the number of examples in the training set.

The training process for a SOM is iterative. Each training iteration involves comparing each member of the dataset to all the neurons in the grid and determining the grid neuron that is closest, in terms of Euclidean distance,

$$d_{pq} = \sqrt{\sum_{i=1}^{n} (x_{pi} - m_{qi})^2} \tag{2.9}$$

to the submitted neuron. The grid neuron that is most similar to the input vector is the winner. Then, the winning neuron and the surrounding neurons are modified, according to this equation:

$$m_i(t+1) = m(t) + h_{ci}(t)[x(t) - m_i(t)] \tag{2.10}$$

where t represents training iterations, m_i represents the winning neuron and x represents the training set member. Here $h_{ci}(t)$ is termed the neighborhood kernel, and it determines which neurons are neighbors and how such neighboring neurons will be modified. Neurons that are further away (in a topological sense) from the winning neuron are modified to a smaller degree than neurons that are closer. The simplest neighborhood kernel is the bubble function[18, 30] (also referred to as a fixed window) which is non-zero for the neighborhood but zero elsewhere. The map in this work implemented a Gaussian kernel,[18] defined as

$$h_{ci}(t) = \alpha(t) \exp\left(-\frac{||r_c - r_i||^2}{2\sigma^2(t)}\right) \tag{2.11}$$

where $\sigma(t)$ is the neighborhood radius at time t which monotonically decreases with time. Thus, the number of neurons considered to be neighbors decreases as training progresses. The term $||r_c - r_i||$ represents the Euclidean distance between the winning neuron and the neighboring neuron. Thus, neighbors closer to the selected neuron will undergo a larger modification than neurons further away from the selected neuron. $\alpha(t)$ is the learning factor, and it influences the extent to which a neuron should be modified. Initially, neurons within a large radius surrounding the selected neuron are considered neighbor neurons. The radius of the neighborhood is decreased in successive training iterations, and in the last stages of training only the nearest neighbors of the selected neuron are modified. The effect of this variable neighborhood function is that in the early stages of training the neurons are modified on a global scale, which leads to a global ordering. Near the end of training, the smaller neighborhood results in fine-tuning of the map features. The neighborhood function thus controls the sensitivity of the map.

The actual modification is controlled by the learning factor, $\alpha(t)$. The learning factor is a function that monotonically decreases from 1 to 0 as training progresses. Once $\alpha(t)$ reaches zero, training stops. Kohonen[18] mentions several ways of modifying $\alpha(t)$, and the implementation used in this work employs a constant decrement,

$$\alpha(t+1) = \alpha(t) - 0.01 \tag{2.12}$$

which implies that after 100 training iterations $\alpha(t)$ will be zero. This represents an upper limit on the number of training iterations.

As mentioned, the result of the training procedure is to create regions of cells on the map that are similar to each other. After training the neurons can be assigned classes by determining which training set member is the closest (in an Euclidean sense) to a given neuron and assigning the class of that training set member to the neuron. Once this is done for all the neurons, a new observation can be classified by assigning the class of the closest neuron in the map, to it. An example of this approach to classification can be found in Chapter 4. An alternative usage is to avoid explicit classification of the neurons in the map and instead cluster the training set members (as well as new observations). In this approach one would simply assign observations to neurons based on Euclidean distance between the members and each neuron. As a result, certain neurons may be assigned more than one training set member and some neurons will not be assigned any. The number of members assigned to a given neuron can be used to compute a density which can then be used to color code the map providing an easy visual display of the topology of the dataset. An example of this approach is shown in Fig. 2.5, which represents a SOM trained using a portion of the NCI AIDS dataset.[31]

2.3 Algorithmic Methods

Breiman[32] categorized a number of statistical methods as algorithmic owing to the fact that they are essentially *model free*. That is, these methods do not help us to understand the relationship between predictor variables and the response by developing a model relationship. However, their utility lies in the fact that they can be used as black box prediction methods and usually show good predictive ability for both classification and regression. There have been a number of applications of algorithmic methods in the physical and medical sciences.[33–36] This class of modeling techniques includes prototype methods such as k-means clustering and learning vector quantization[37] as well ensemble methods such as the random forest technique. In this section we describe two algorithmic techniques used in this work.

2.3.1 Random Forests

The random forest (RF) method was developed by Breiman as an extension of the decision tree[38] technique. Decision trees are non-parametric, nonlinear models that allow the user to easily understand how or why an observation is classified or predicted. They have been extensively used in the medical field[39, 40] as well as in various chemical[41–43] and biological[44, 45] applications.

The goal of the decision tree technique is to split the dataset into a tree-like structure, using a single descriptor at each split point. A variety of algorithms to achieve this are available and we focus on the recursive partitioning algorithm.[46] The dataset (also known as the root node), D, is first split into two nodes, say D_1 and D_2, based on the value of a selected descriptor, say X_i. If X_i is binary in character then the j^{th} observation from D is placed in D_1 or D_2 depending on whether the value of X_i for the j^{th} observation, denoted by X_{ij}, is 0 or 1. In case X_i is real valued, a specific value of X_i, say x_i is calculated such that if $X_{ij} < x_i$ then the j^{th} observation goes into D_1 or goes into D_2 otherwise. The method by which a descriptor is selected is based on a quantity known as *purity*. All the available descriptors are considered individually as candidates for the splitting decision. The purity of a split can be defined in a number of ways. One possible approach is to define the purity of a split as the fraction of observations, in the resultant nodes, that will be of a single class. The descriptor that leads to the highest value of purity will be selected to perform the splitting. Other definitions of the purity include the Gini index, χ^2 and G^2.[38] It should be noted that the same descriptor can be chosen at multiple split points.

After all the observations have been placed in either one of the nodes, the algorithm considers each node and performs the same operation described above. Thus D_1 would be split into two nodes, say D_3 and D_4 and similarly for D_2. A flowchart summarizing the algorithm is shown in Fig. 2.6. If this process is repeated continously, the end result will be a *perfect tree*, where each node contains a single observation. Nodes which cannot be split further are termed leaf nodes. Such a tree will perfectly predict the data used to build the tree but will be nearly useless for new observations. Thus a perfect tree will overfit the data. As a result, heuristics are used to determine when the tree should stop growing. These include specifying a minimum node size, such that nodes with fewer observations will not be split further. Alternatively, a node is not split, if it exhibits a purity greater than a certain specified value or the purity does not increase as a result of the split. Other possibilities include a variety of cross-validation methods. These heuristics are collectively known as pruning rules and detailed discussions of this area can be found in the statistical literature.[39, 46, 47] An example of a tree is shown in Fig. 2.7. The figure represents a decision tree for a hypothetical classification problem. Three descriptors were available at each split point and the fractional purity measure was used to select a descriptor at each split. As can be seen, this measure results in each node, generated by a split, consisting mainly of a single class.

After a tree of the required size (i.e., number of nodes) has been created it can then be used for predictive purposes by determining in which leaf node a new observation would belong, and assigning a class based on the majority class of the leaf node or calculating a property value, by averaging the property values of the observations contained in the leaf node.

The philosophy behind the the RF method is that the technique is able to provide high predictive ability by averaging the predictions of a large number of individual decision trees. In other words, the RF technique is an ensemble method based on *forests* of decision trees. The technique is applicable to both classification and regression. The following discussion presents the fundamentals of the RF technique and its use in providing a measure of variable importance.

Since the RF method is based on decision trees, the features of the latter that make it an attractive option in the development of predictive models also apply to RF's. These features include the ability to handle high-dimensional data, the ability to ignore irrelevant variables (thus obviating the need of feature selection) and the ability to provide some measure of interpretability. However one of the major drawbacks of the decision tree method is its low predictive ability. Random forests, as an extension of the decision tree method, exhibit a higher predictive ability coupled with other features such as an internal measure of accuracy and a measure of variable importance.

Random forests are closely related to other tree based ensemble techniques such as bagging,[48] boosting[49] and random split selection.[50] The method consists of generating an ensemble of N trees denoted by[51]

$$R = \{T_1(X), T_2(X), \ldots, T_N(X)\} \tag{2.13}$$

where X is defined as a p-dimensional vector of descriptors. The output of this ensemble is be denoted by

$$\{\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_N\} \tag{2.14}$$

where $\hat{Y}_i$ is the output of the i^{th} tree. The predicted value reported by the random forest, $\hat{Y}$, is then the majority vote of the N outputs (for classification) or the mean of the N output values.

The algorithm to build the forest can be described as follows. Consider a training set, D, of n observations and p independent variables (represented by p-dimensional vectors). The observations can be categorical (for classification) or real valued (for regression). A random subset of the training data is selected with replacement and a decision tree is built using this *bootstrap* sample. The evolution of the decision tree is slightly different from the original algorithm described by Breiman[38] in that, at each node m $(1 \leq m \leq p)$ descriptors are selected randomly from the descriptor pool. When $m = 1$ this is equivalent to random splitting and when $m = p$ this is equivalent to bagging.[48] Using this rule to split nodes, a tree is grown to its maximal size and pruning is not carried out. Another bootstrap sample is selected and another tree is grown. This procedure is repeated till the requisite number of trees (N) have been grown. Fig. 2.8 summarizes the steps involved in the creation of a random forest model as a flowchart.

The fact that subsets of the dataset are used to build the forest allows us to use the observations not used during building (termed out-of-bag or OOB observations) to obtain a measure of predictive performance. This measure is termed the out-of-bag estimate[52] and can be considered a parallel cross validation since it is estimated for each training step. The OOB estimate is obtained by considering the OOB part of the data for the i$^{\text{th}}$ tree, denoted by D_i^{OOB}. The i^{th} tree is used to predict the property of the observations in D_i^{OOB}. It has been shown[36] that on average each tree uses approximately $1 - e^{-1} = 2/3$ of the whole dataset and hence the size of D_i^{OOB}, is on average 1/3 of the dataset. This implies that each observation will be in the OOB data about 1/3 of the time. Consequently, the OOB estimates can be aggregated to provide an ensemble prediction for each observation. In the case of regression, this result is an out-of-bag estimate of the mean square error (MSE) that can be used to approximate the MSE for the entire ensemble of trees and can be written as[51]

$$MSE \approx MSE^{OOB} = \frac{1}{n}\sum_{i=1}^{n}\left[\hat{Y}^{OOB}(X_i) - Y_i\right]^2 \tag{2.15}$$

It has been shown[51] that the MSE obtained by Eq 2.15 agrees well with the results of a k-fold cross validation scheme.

In addition to obtaining performance estimates of the random forest, the OOB dataset also allows us to obtain a measure of importance of the descriptors in the pool supplied to the algorithm. The measure is obtained by first calculating the predicted values for the OOB data for a given tree. Next, each descriptor in the OOB dataset is individually scrambled and predictions are made on the scrambled OOB dataset, for each scrambled descriptor. This procedure is repeated for all the trees grown in the forest. After training is complete, the overall OOB estimates (MSE) for the unscrambled and scrambled sets can be evaluated using Eq 2.15. The importance of the j$^{\text{th}}$ descriptor is then given by

$$\text{Importance}_j = MSE - MSE_j \tag{2.16}$$

where MSE represents the OOB estimate for the unscrambled data and MSE_j represents the OOB estimate for the datasets in which the j^{th} descriptor was scrambled. This procedure allows the ranking of the descriptor used in the random forest in order of relative importance. Descriptors that play a more important role in the predictive ability of the model will have a higher value of importance compared to descriptors that play an insignificant role. The importance measure for each descriptor can be plotted to allow easy visual inspection as shown in Fig. 2.9. From this figure it is clear that SURR-5 is significantly more important than the other descriptors owing to its large separation on the X axis. These plots are used in Chapter 7 to provide a comparison with interpretation schemes for other types of models.

2.3.2 *k*-Nearest Neighbor Algorithm

The k-nearest neighbor (kNN) method is very simplistic in nature and assumes that observations that are close in the space of the predictor variables will be close to each other in the space of the response variable. This method can be applied to both regression as well as classification problems, though when faced with high-dimensional data, kNN regression does not perform very well.[36]

In the case of regression the kNN fit for the i^{th} observation is defined as

$$\hat{Y}(x_i) = \frac{1}{k} \sum_{x_j \in N_k(x)} y_j \tag{2.17}$$

where $N_k(x)$ is the set of k points closest to x_i, that is, the neighborhood of x_i. Eq. 2.17 simply averages the observed values of the k nearest neighbors of x_i to obtain the predicted response for this observation. From the above discussion, the first step of the kNN method is to obtain the neighborhood for a given observation. This implies the choice of a distance metric. The most common metric used is the Euclidean distance, defined as

$$d_{ij} = ||x_i - x_j|| \tag{2.18}$$

where x_i and x_j are the independent variables for the query observation and prospective neighbor respectively. Other possibilities include the Manhattan distance and Mahalanobis distance though the choice of distance metric does not appear to affect the results significantly.[53]

In the case of kNN classification, the class of a query observation is simply the majority class of its nearest neighbors. Fig. 2.10 shows a schematic diagram of the working of the kNN algorithm. In the figure, the central white point is the query point and the points connected to it correspond to its three nearest neighbors. In the case of kNN regression, the property of the query point would be the average of the property of the nearest neighbors. In the case of kNN classification, the class of the query point would be the majority class of the nearest neighbors and in this case, the query point would be classified as *blue*. As opposed to kNN regression, kNN classification performs reasonably well when faced with high-dimensional data. This is due to the trade off between bias and variance. It has been shown[54] that in the case of a 1-nearest neighbor classifier the asymptotic error rate is never more than twice the Bayes error rate. It is evident that this observation considers the asymptotic region and hence assumes that the bias of the nearest neighbor rule is zero. In real problems (especially high-dimensional cases) this is not always the case and the bias term can be substantial. Nearest neighbor classification implicitly assumes that the class probabilities are approximately uniform within the neighborhood of the query point. In the case of high-dimensional datasets, this assumption does not necessarily hold and in fact, when the number of dimensions is high the class probabilities might vary significantly in a certain direction. One approach to alleviating this problem is the use of adaptive metrics which modify the distance metric so that class probabilities in the resultant neighborhoods do not vary significantly. Examples of such adaptive nearest neighbor methods include that proposed by Friedman[55] and the discriminant adaptive nearest neighbor (DANN) rule described by Hastie et al.[56]

Given a suitable distance metric a kNN algorithm only requires that a suitable value of k be chosen. In many cases setting k to 1 provides reasonably good predictive performance for classification purposes. In general, optimal values of k are obtained via trial and error. A more systematic approach is to use a cross-validation scheme to obtain the best value of k for a given dataset. One example of this approach has been described by Shen et al.[57] in which the value of k along with the selection of variables used for classification were optimized simultaneously.

2.4 Optimization Methods

Optimization is a fundamental topic in QSAR modeling. In this context, there are two main applications of optimization techniques. The first involves the optimization of parameters for a model such as the weights and biases in a feed-forward neural network described previously. The other area where optimization plays an important role is in the selection process. Here, selection can mean the selection of compounds from a library or design of a library from various components[58–60] or it can mean the selection of descriptors to build models with. In this section we concentrate on the latter form of selection. As will be described in Chapter 3, the model building process begins with the generation of a large number of descriptors. In the interests of parsimony, our goal is to use the minimum number of descriptors to develop a good predictive model. Thus, we must select *good* subsets of descriptors. The definition of *good* depends on the modeling technique used to build the models after descriptor selection and will be discussed in the following sections. Though statistical methods exist to perform descriptor selection (or variable selection as it is termed in the statistical literature) such as stepwise regression, backward elimination and forward selection, these methods are generally restricted to linear regression models. However, a more important reason for avoiding these methods is due to a number of inherent drawbacks which include falsely narrow confidence intervals,[61] incorrect p-values, biased regression coefficients that require shrinkage,[62] severe problems with collinearity[63] and so on. Furthermore backward or forward selection algorithms by their nature will ignore certain combinations of variables (since in the former case variables removed from consideration are not considered again and in the latter case variables are based on the current subset that has already been selected). Owing to these restrictions we avoid statistical selection algorithms and instead focus on optimization algorithms to carry out descriptor selection.

Optimization methods can be divided into two broad classes: deterministic and stochastic. Examples of deterministic methods include the BFGS algorithm[1–4] and the Nelder-Mead simplex algorithm.[5] Examples of stochastic methods include the genetic algorithm[6] and the simulated annealing algorithm.[9] The choice of method largely depends on the nature of the solution space. Deterministic methods are preferred in cases where there is known to be one global minimum in the solution space and when the dimensionality of the solution space is relatively small. Multiple local minima can exist and various improvements to the standard algorithms are available to overcome this situation. When the solution space is very large (or possibly combinatorial in nature) and may have multiple minima but no distinct global minimum, a stochastic algorithm which is able to effectively *sample* the solution space is the preferred option. This does not imply that a stochastic method cannot be used in the former case. Genetic algorithms have been used to optimize the weights and biases in neural networks.[64,65] In the various applications discussed in this work parameter optimization has been carried out using deterministic methods whereas descriptor selection has been carried out using stochastic methods. The following sections describe the principles underlying the genetic algorithm and simulated annealing and details of their implementations.

2.4.1 Genetic Algorithms

A genetic algorithm is a member of the class of optimization algorithms known as evolutionary algorithms, which utilize the concepts of biological evolution to develop efficient optimization strategies.[8] GA's have been used widely in the field of QSAR modeling,[66–68] cheminformatics[59,69,70] and chemometrics.[71–74] The application of genetic algorithms in this work are focused on their use as efficient tools to search large dimensional spaces. More specifically, one application of GA's in QSAR modeling is to search a descriptor space to find optimal subsets of descriptors that can be used to build predictive models. Fig. 2.11 shows a flowchart of the generic genetic algorithm and this section describes the steps in detail.

As mentioned above, a GA is based on the principles of evolution. As a result much of the terminology from the field of biological evolution has been adapted for use in the field of genetic algorithms. Thus we define an individual as consisting of a chromosome and an associated fitness value. When using a GA for descriptor selection, the chromosome is simply a subset of descriptors (of user specified length) chosen from the descriptor pool that is being searched. A population is defined as a collection of individuals. The first step of the GA is to initialize the population. This is achieved by randomly generating a user specified number (usually 40 to 50) of descriptor subsets of user specified size. Each descriptor subset is used to build a model (which can be a linear regression model or a CNN model). The root mean square error (RMSE) for each model is used to determine the fitness of the individual. The implementation used in this work does not use the raw RMSE value but instead uses a linearly scaled form. The actual form of the fitness function depends on the nature of the model to be developed. For linear models the fitness for the i^{th} individual in the population is defined as

$$\text{Fitness}_i = \left(2 - \frac{\text{RMSE}_i}{\text{RMSE}_{\text{avg}}}\right)^{-1} \tag{2.19}$$

where RMSE_i is the RMSE for the i^{th} individual and RMSE_{avg} is the average RMSE for the whole population. In the case of CNN models, the fitness function is defined by the cost function described in Section 2.2.1. Once the fitness for each individual has been evaluated, the population is ranked.

The next step is to create a child population. First a mating list is created, which is of the same size as the current population. Those individuals with fitness greater than the population average (which from Eq. 2.19 is greater than 1.0) are automatically placed in the mating list. By definition, this will fill up half of the available slots. The remaining slots in the mating list are filled by using a roulette wheel selection procedure[6] to select individuals from the current population. Once the mating list is created a child population is then generated by successively selecting two individuals from the mating list at random and applying genetic operations.

The first operation is termed crossover, and involves the the swapping of portions of the chromosomes of a pair of individuals. The GA literature describes a number of variations of the crossover operation.[6] The current implementation restricts itself to the single point crossover. In this type of crossover a split point is chosen in the descriptor subset. Then the descriptors from one side of the split point in the two individuals are swapped to give rise to two new individuals. This operation is shown graphically in Fig. 2.12. The figure represents a crossover performed on two individuals having a chromosome (descriptor subset) of length 5. The split point is chosen at the fourth descriptor and the descriptors on the left of the split point are swapped resulting in two new individuals. The goal of crossover is to generate new individuals that will have the good features of the parent individuals. That is, if two individuals have a high fitness this implies that certain parts of their chromosomes (i.e., certain descriptors) are responsible for their fitness. By combining a portion of the chromosomes of two fit individuals, we expect that the children will exhibit equal if not better fitness.

The second genetic operation is termed mutation and is performed on a single child individual. It should be noted that mutation is not performed on all individuals in a population but is carried out only 5% of the time, mirroring the low frequency of mutation in biological evolution. In a genetic algorithm the mutation operation is performed by randomly changing a part of the chromosome of an individual. That is, a random descriptor within an individual is replaced with a randomly chosen descriptor from the descriptor pool. This is shown schematically in Fig. 2.13. The goal of the mutation operation is two-fold. First, random mutations prevent the algorithm from getting stuck in a local minimum and second, mutations prevent the phenonemon of premature convergence. This occurs when the algorithm creates very similar (or even identical) individuals whose fitness is high, but not necessarily optimal. The mutation operation can also be viewed as a method to maintain diversity within a population, though this does not entirely solve the problem of premature convergence as noted by Goldberg.[6]

With the application of these two operations we end up with a second, child, population. The fitness of the individuals in this population are evaluated and the individuals ranked. The second generation population is then created by randomly selecting individuals from the the top 50% of the previous population and the child population. Finally, if the best model in the child population is of lower fitness than the best model from the previous population, the best model from the previous population is kept in the second generation. With the formation of the second generation population, the whole process is repeated. This continues for a user specified number of cycles (usually 1000) and at the end the top ranked individuals (i.e., the top ranked descriptor subsets and associated RMSE values) are reported to the user.

2.4.2 Simulated Annealing

Simulated annealing is a generalization of the Metropolis Monte Carlo method[75] to optimization problems. The original method was devised as an efficient way to evaluate the Boltzmann average of a given atomic or molecular property. The method was extended by Kirkpatrick et al.[9] to determine the most stable state of a system. This modification was based on the physical phenonemon of annealing in which a melt (such as a glass or metal) is initially at a high temperature and then allowed to cool slowly, such that at any time, the melt is in thermal equilibrium. As the temperature is decreased the atoms in the melt will achieve increasingly ordered states and when the final temperature (say, room temperature) is reached the configuration of the atoms should be that of the most stable state.

This modification of the Metropolis method is easily extendable to combinatorial optimization problems and more specifically for QSAR, feature selection. In terms of an optimization problem, the temperature term in the simulated annealing algorithm effectively controls the size of the solution space and the cooling schedule narrows the space over time, allowing the algorithm to reach the global minimum of the solution space.

The original algorithm described by Metropolis et al. considered an initial thermodynamic configuration of a system with energy E at a temperature T. This configuration was perturbed and the change in energy, ΔE, was evaluated. If the change in energy was negative the new configuration was accepted and if positive, the configuration was accepted with a probability equal to the Boltzmann factor, $\exp(-\Delta E/kT)$. This procedure was repeated a number of times to obtain sampling statistics and then the temperature was reduced by a small amount. The whole procedure was then repeated till the final temperature was achieved. In terms of a feature selection problem, the thermodynamic configuration is replaced by a set of descriptors and the energy is replaced by a cost function which in the case of the studies presented in this work is either a linear regression routine or CNN routine. Thus, the algorithm starts out with a random descriptor subset (configuration), say x_0, selected from the descriptor pool and the value of the cost function (which is the RMSE for linear models and Eq. 2.6 for CNN models) for this descriptor subset is calculated, say $C(x_0)$. Next, the descriptor subset is perturbed by randomly replacing a single descriptor by a randomly selected descriptor from the pool. The value of the cost function for this new configuration is then determined, say $C(x)$. If $C(x) < C(x_0)$ the new configuration replaces the previous one and we repeat the perturbation process. If $C(x) \geq C(x_0)$ then a detrimental step has been taken. The new configuration is accepted with a probability equal to the Boltzmann acceptance probability, P. If the new configuration is still not accepted, the algorithm replaces the new configuration with the old one and returns to the perturbation step. The working of the algorithm is shown schematically in Fig. 2.14. The whole procedure results in a single, low cost descriptor subset. The algorithm is then repeated with a different random descriptor subset to build up a pool of low cost descriptor subsets which can then be investigated in more detail.

The simulated annealing algorithm used in this work is an implementation of generalized simulated annealing described by Bohachevsky.[76,77] This method differs from the classical simulated annealing algorithm by introducing a step size Δr and a normalized n-D vector v. The vector v corresponds to a set of random perturbations of the current configuration x. v is obtained by generating a set of random numbers from $N(0,1)$ denoted by $u_i, i = 1 \ldots n$ and evaluating

$$v_i = \frac{u_i}{\sum_{i=1}^{n} u_i^2} \tag{2.20}$$

The new configuration y is then given by

$$y = x + \Delta r v \tag{2.21}$$

which lies in the neighborhood of x. The second modification is that the acceptance probability P, approaches zero as the configuration approaches the global optimum. This gives P the form

$$P = \exp\left(-\beta \frac{C(y) - C(x)}{C(x) - C_{est}}\right) \tag{2.22}$$

where $C(x)$ and $C(y)$ are the values of the cost function for the configurations x and y and C_{est} is the estimated global optimum. β is a parameter that controls the cooling schedule and corresponds to the effective temperature term, kT, in the Boltzmann factor. The choice of β is important as too small a value will result in a random walk and too large a value will cause the algorithm to converge to a local minimum. β begins with a small value which allows detrimental steps to be taken relatively frequently. This allows the algorithm to explore a larger space. As the algorithm progresses, β is increased making the probability of acceptance of a detrimental step lower, thus shrinking the search space. To determine the initial value of β the algorithm is run for a set number of iterations and β is adjusted until the relation $0.5 < \bar{P} < 0.9$ (where $\bar{P}$ is the mean probability of a detrimental step) is achieved. In the case of the current implementation, the algorithm is allowed to run for 1000 iterations and for each detrimental step, the equation

$$P = \exp\left(-\beta \Delta C\right) \tag{2.23}$$

is solved assuming $P = 0.8$. Here ΔC is the difference in the value of the cost function for the detrimental configuration and the previous configuration. At the end of the iterations the average value of β is taken as the starting value for the actual run. To prevent premature convergence, β is multiplied by 2 every 100 iterations for a maximum of 50000 iterations. If detrimental steps occur more than 900 times in a row β is reset to the starting value and if this occurs twice in a row the algorithm exits.

2.5 Conclusions

In this section I have presented the underlying details of the modeling and optimization algorithms used in this work. For each class of modeling technique or optimization technique described here, there is a number of alternatives that have not been discussed. These include variants of the fundamental neural network model such as the probabilistic neural network and various types of linear modeling techniques such as ridge regression, linear discriminant analysis and so on. The QSAR literature has numerous applications of these and other modeling techniques. The field of optimization is certainly much more detailed than has been described in this chapter and, the literature describes numerous algorithms and variants that are suited for both general use as well as for special cases. However, the focus of this work is not on the modeling or optimization techniques themselves. Rather, the goal of this work is to develop and implement techniques that allow us to obtain meaningful knowledge from data using predictive or descriptive models. The methods described in this chapter allow us to achieve the first step, namely, the development of the model itself. Subsequent chapters describe various methods that have been developed to ensure validity and provide interpretability.

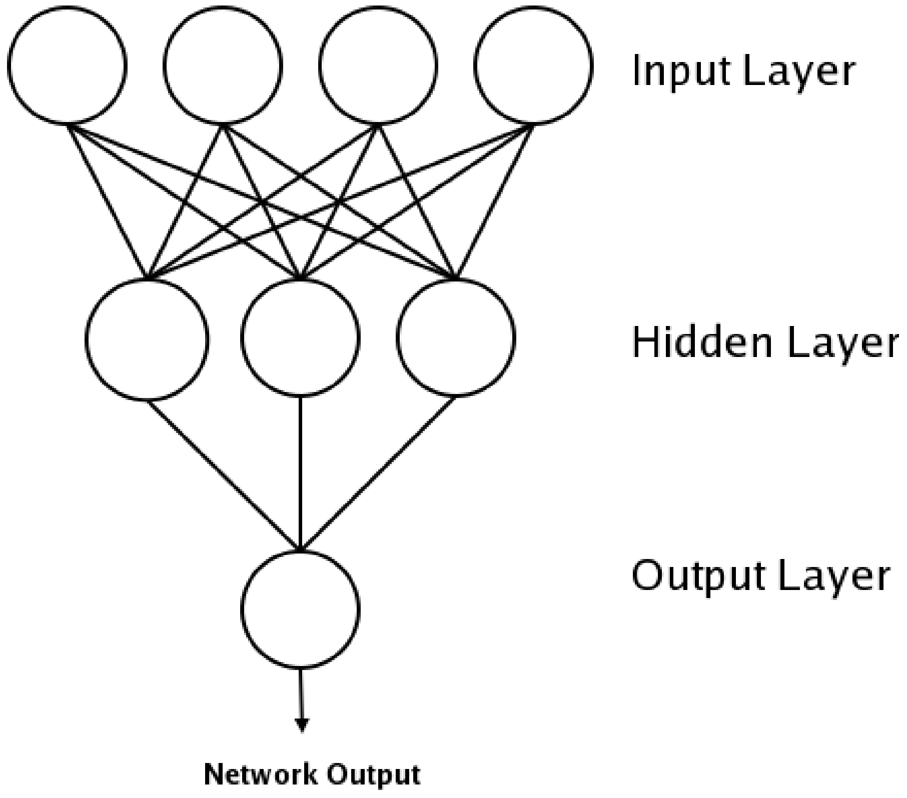

Fig. 2.1. A schematic diagram of a 3-layer, fully connected feed-forward neural network

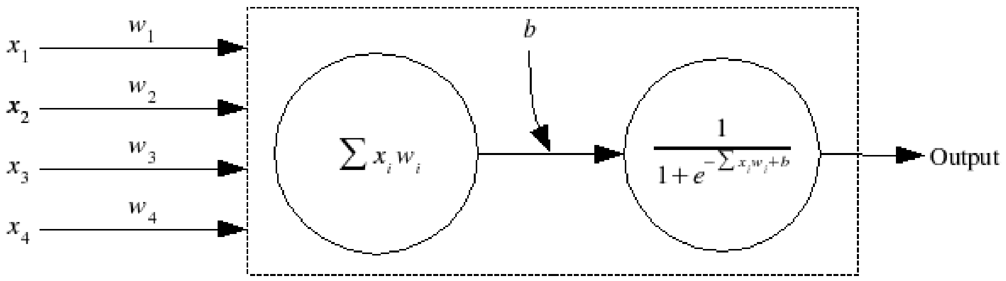

Fig. 2.2. A more detailed view of a single hidden layer neuron. The x_i's represent the output value of the neurons in the preceding layer and w_i's correspond to the weights for the connections between this neuron and those in the preceding layer. b represents the bias term for this neuron.

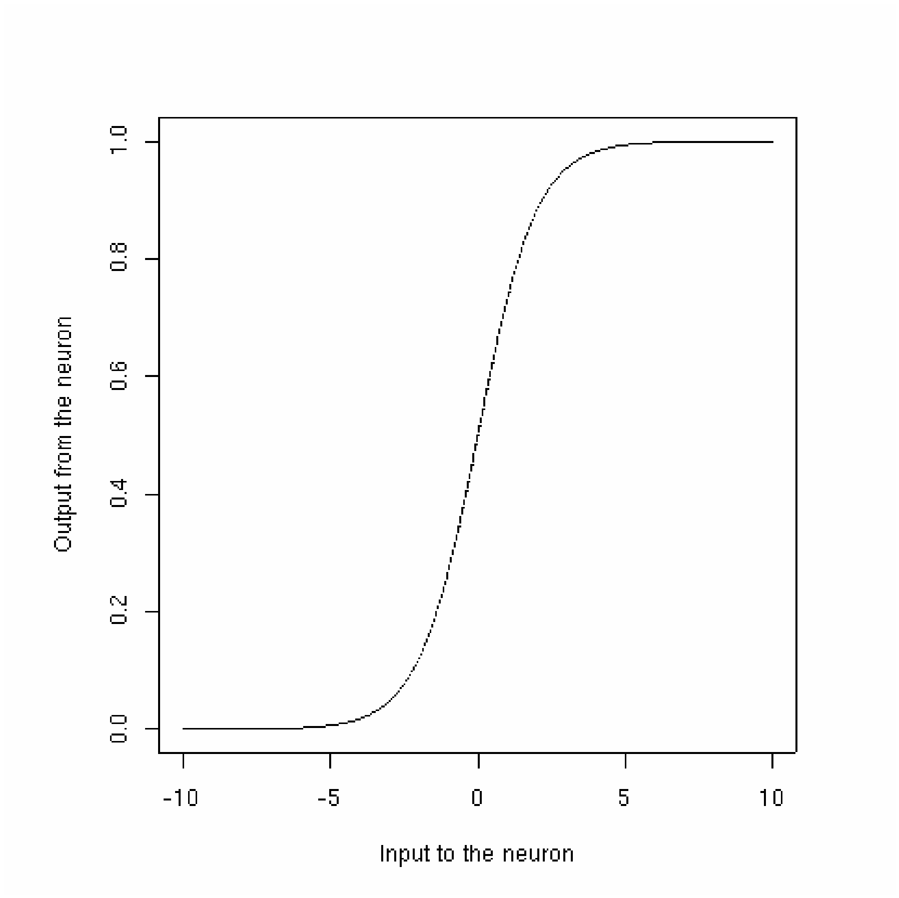

Fig. 2.3. A plot of the signmoidal transfer function used in the implementation of the neural network algorithm in this work

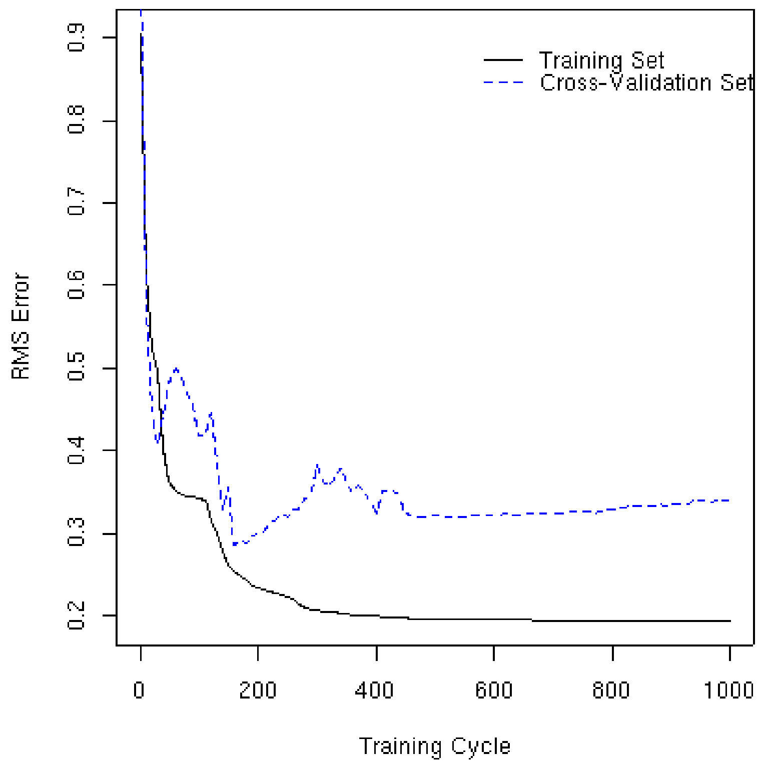

Fig. 2.4. A plot showing the variation of training set and cross-validation set RMSE with training cycle. The global minimum of the cross-validation curve indicates the training cycle at which the optimal weights and biases occur

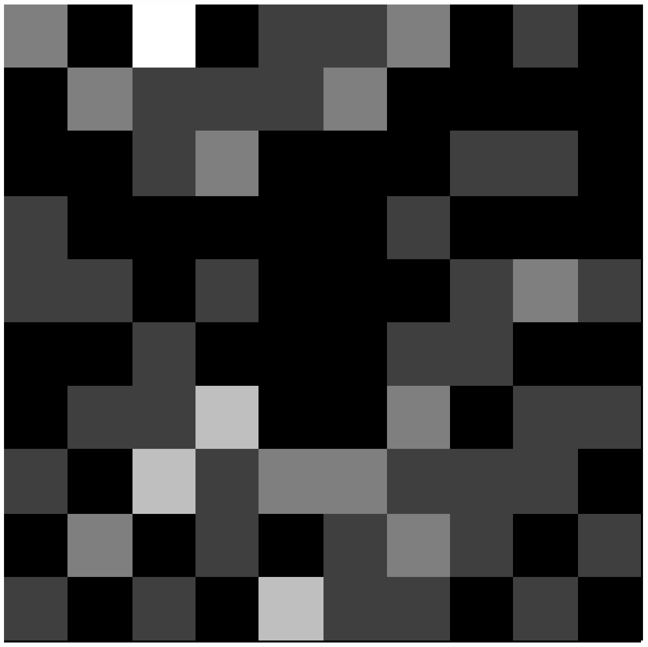

Fig. 2.5. A clustering of the first 5000 compounds from the NCI AIDS test dataset[31] obtained using a SOM. The grid dimensions are 10×10 and the neurons are color coded based on the number of compounds that map to them. Thus black neurons have no members from the training set mapped to them whereas the white neurons have the maximum number of observations mapped to them.

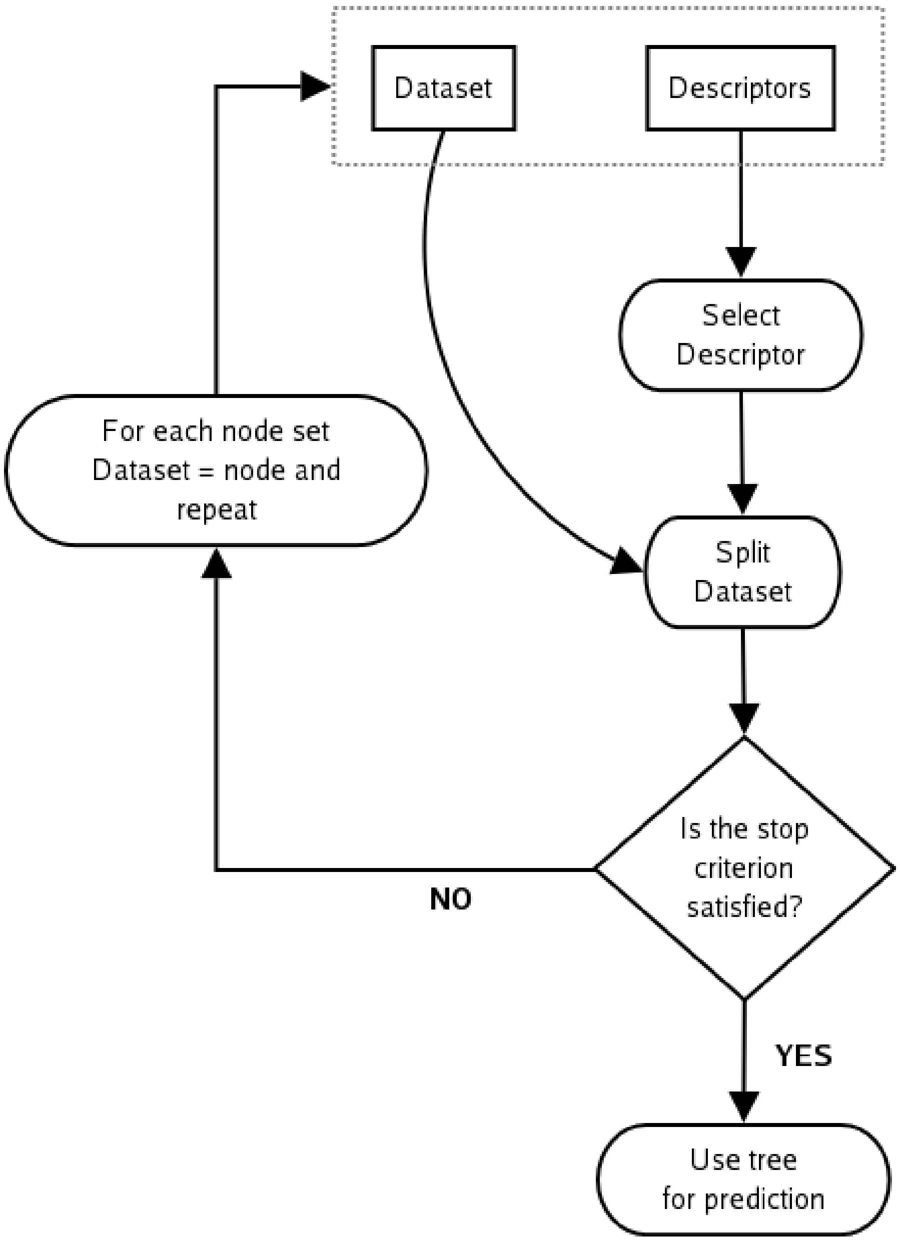

Fig. 2.6. A flowchart illustrating the recursive partitioning algorithm used to generate a decision tree

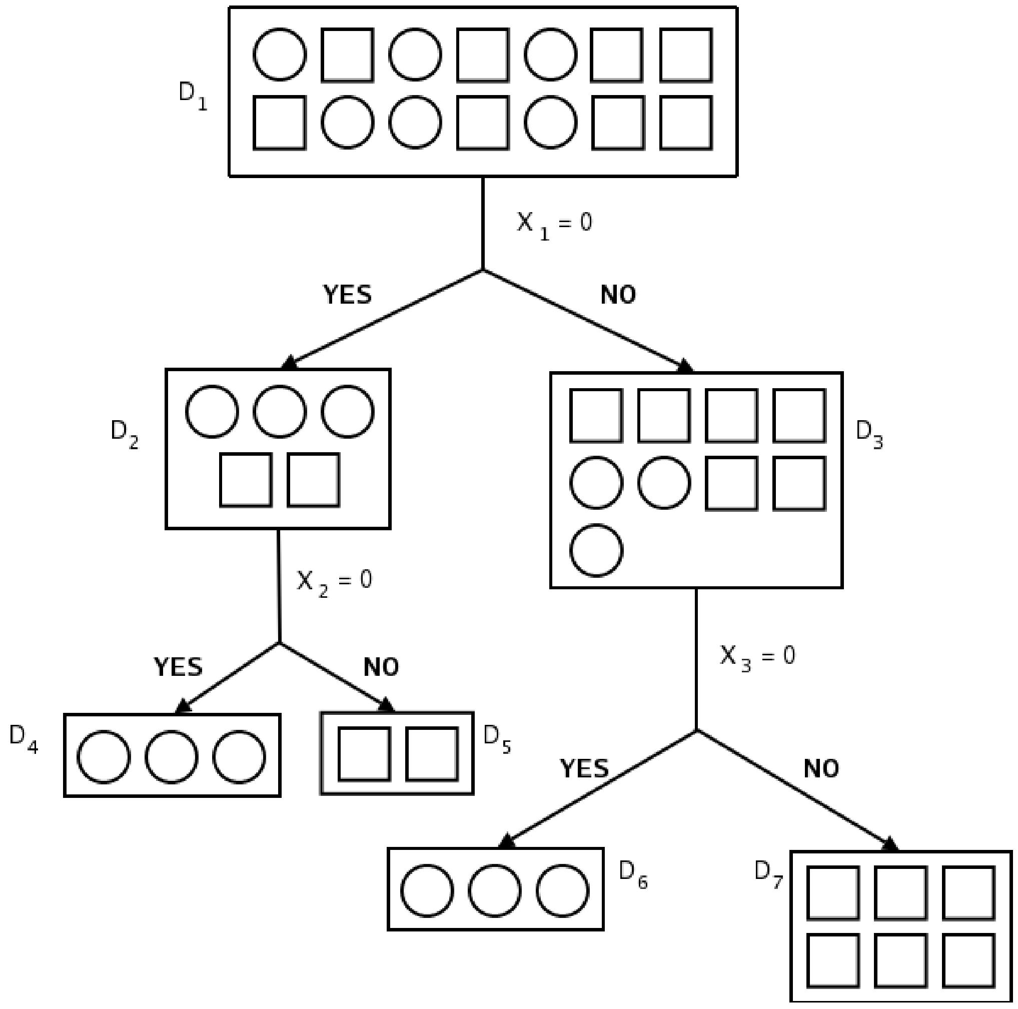

Fig. 2.7. A schematic diagram of a decision tree for a hypothetical clasification problem. The purity measure used to grow the tree was defined to be the fraction of a single class in the nodes created by a prospective split. Three binary descriptors, X_1, X_2 and X_3 were available for splitting and the D_i's correspond to each node. Nodes D_4, D_5, D_6 and D_7 represent leaf nodes.

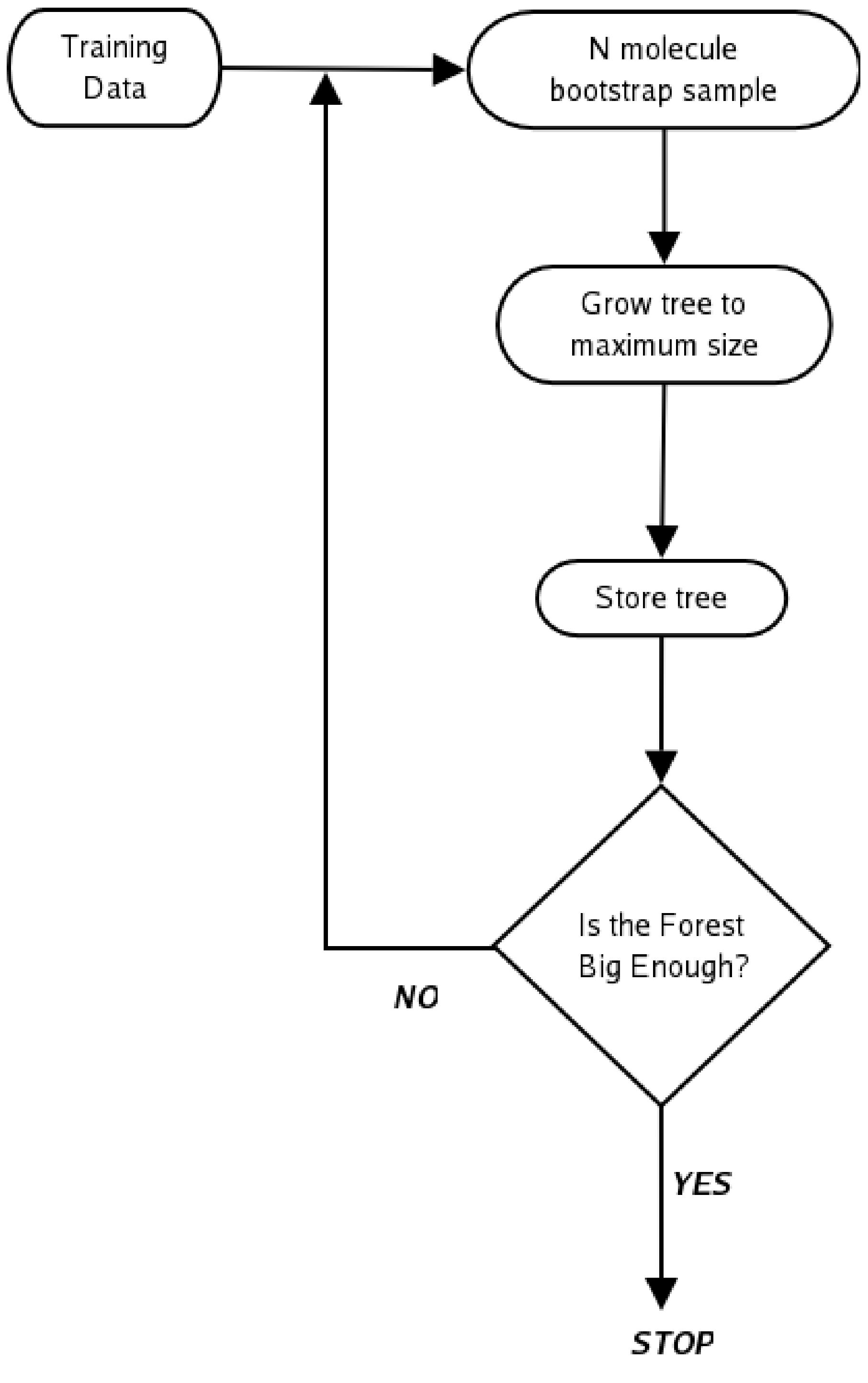
Training Data
N molecule bootstrap sample
Grow tree to maximum size
Store tree
Is the Forest Big Enough?
NO
YES
STOP

Fig. 2.8. A flow chart describing the working of the random forest algorithm

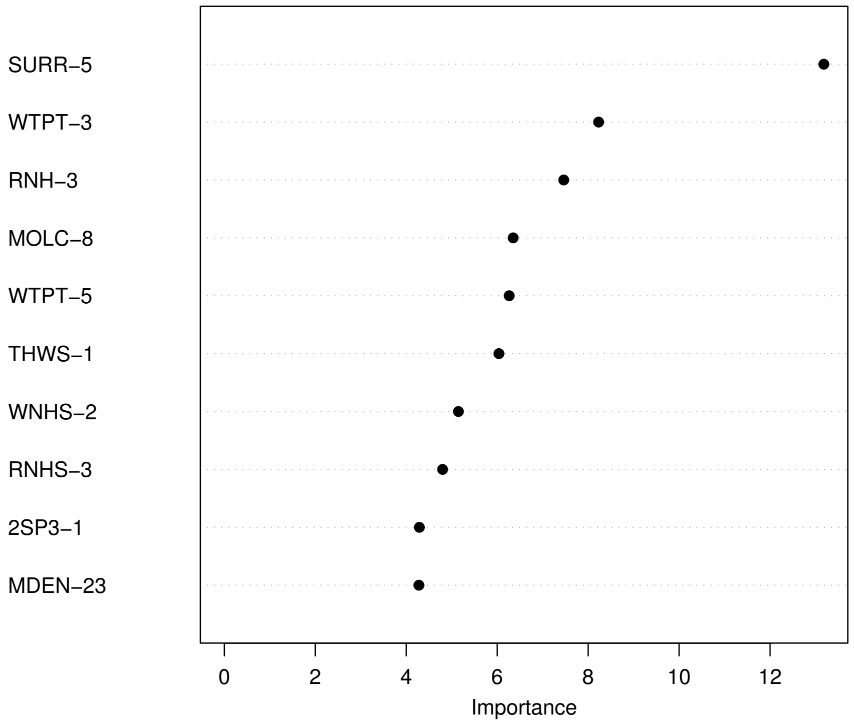

Fig. 2.9. A random forest descriptor importance plot

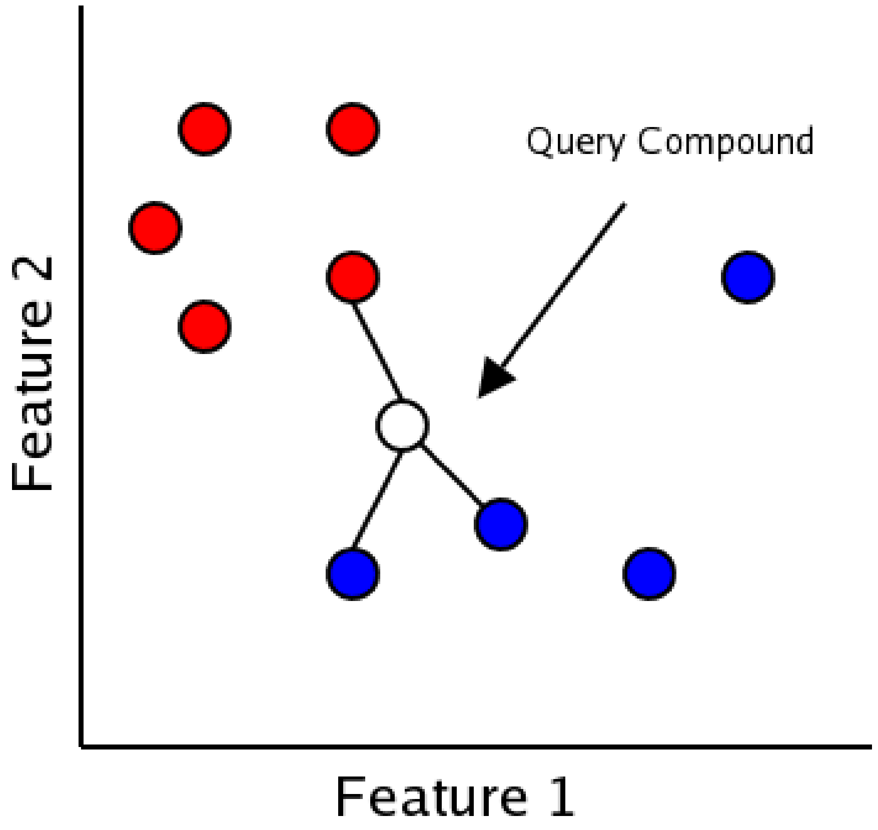

Fig. 2.10. A schematic diagram of the kNN algorithm

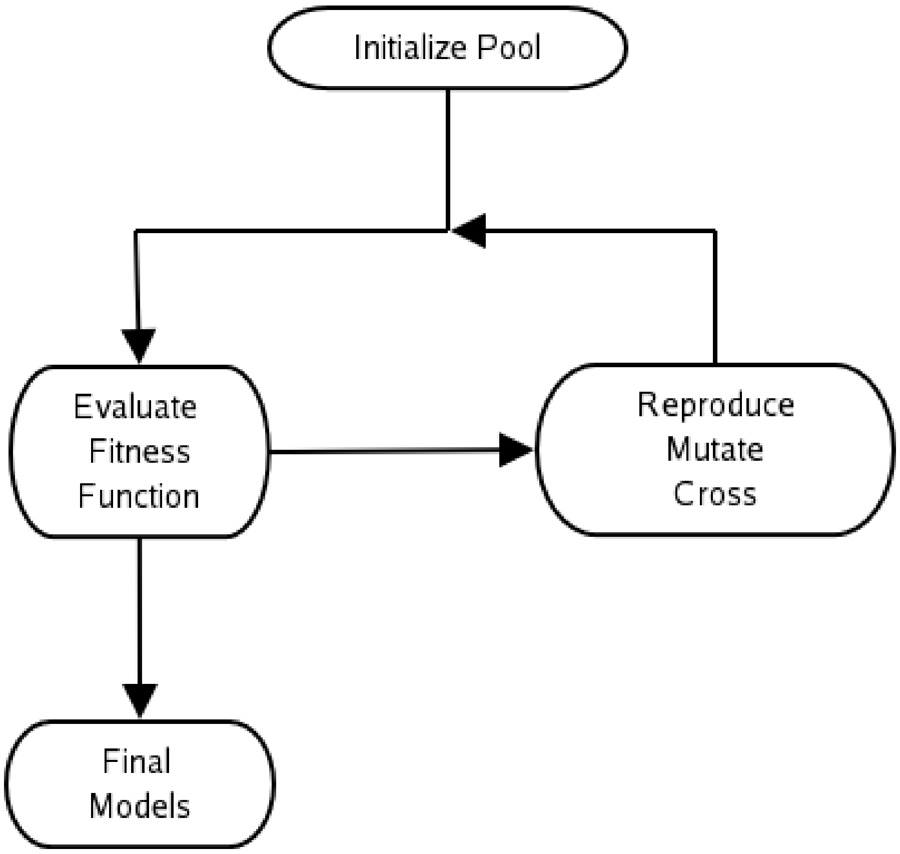

Fig. 2.11. A flow chart describing the working of a genetic algorithm

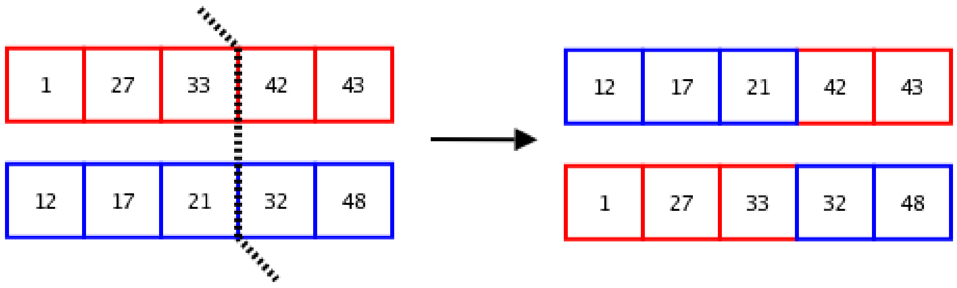

Fig. 2.12. A schematic diagram of the single point crossover operation. The grids on the left represent the parents and the grids on the right represent the children formed after crossover. The portion of the chromosomes to the left of the split point are swapped.

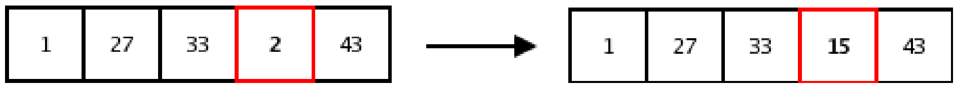

Fig. 2.13. A schematic diagram of the mutation operation

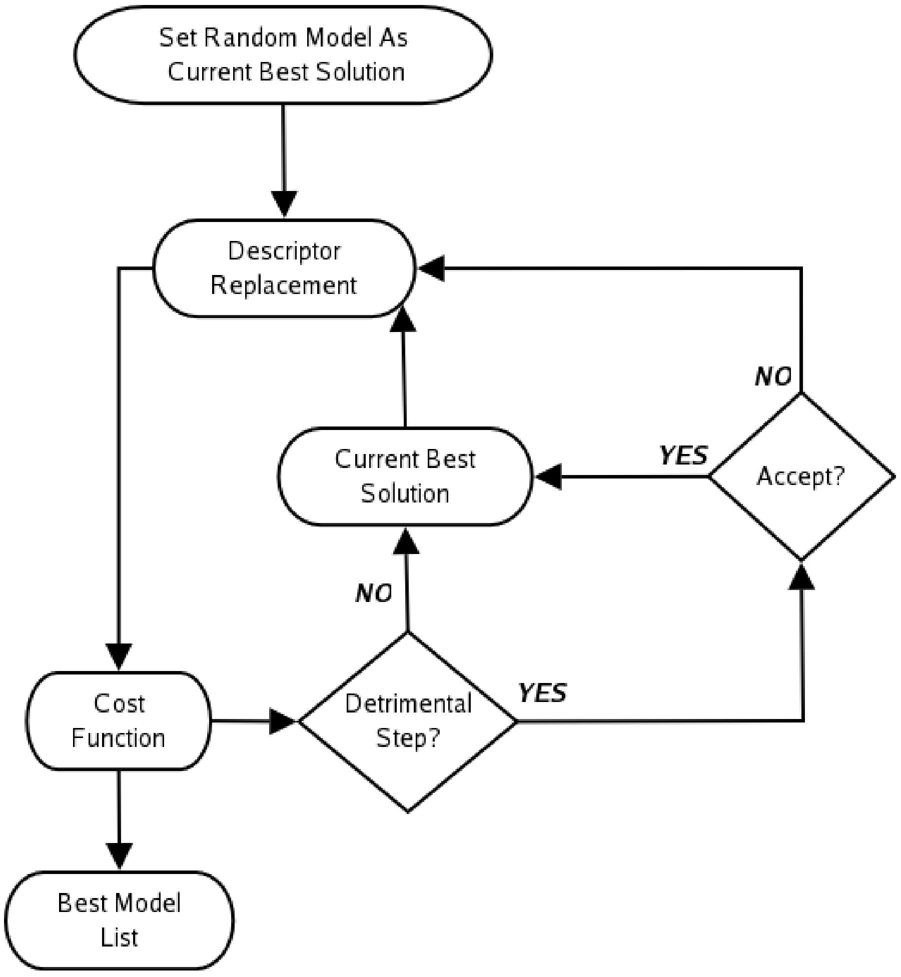

Fig. 2.14. A flow chart describing the working of a simulated annealing algorithm

References

[1] Broyden, C. The Convergence of a Class of Double-Rank Minimization Algorithms 2, The New Algorithm. *J. Inst. Math. Applic.* **1970,** *6,* 222–231.

[2] Fletcher, R. A New Approach to Variable-Metric Algorithms. *Comput. J.* **1970,** *13,* 317–322.

[3] Goldfarb, D. A Family of Variable-Metric Algorithms Derived by Variational Means. *Math. Comput.* **1970,** *24,* 23–26.

[4] Shanno, D. Conditioning of Quasi-Newton Methods for Function Minimization. *Math. Comput.* **1970,** *24,* 647–656.

[5] Nelder, J.; Mead, R. A Simplex Method for Function Minimization. *Comput. J.* **1965,** *7,* 308–315.

[6] Goldberg, D. *Genetic Algorithms in Search, Optimization & Machine Learning;* Addison-Wesley: Reading, MA, 2000.

[7] Holland, J. Genetic Algorithms. *Sci. Am.* **1992,** *267,* 66–72.

[8] Forrest, S. Genetic Algorithms: Principles of Natural Selection Applied to Computation. *Science* **1993,** *261,* 872–878.

[9] Kirkpatrick, S.; Gelatt, J. C.; Vecchi, M. Optimization by Simulated Annealing. *Science* **1983,** *220,* 671–680.

[10] Norinder, U. Single and Domain Mode Variable Selection in 3D QSAR Applications. *J. Chemom.* **1996,** *10,* 95–105.

[11] Golbraikh, A.; Tropsha, A. Beware of q^2. *J. Mol. Graph. Model.* **2002,** *20,* 269–276.

[12] Neter, J. *Applied Statistics;* Allyn and Bacon Inc.: Boston, MA, 1988.

[13] Barnett, V. Probability Plotting Methods and Order Statistics. *Appl. Stat.* **1975,** *24,* 95–108.

[14] Rousseeuw, P.; Leroy, A. *Robust Regression and Outlier Detection;* Wiley Series in Probability and Mathematical Statistics John Wiley & Sons: Hertfordshire, England, 1987.

[15] Gupta, A.; Park, S.; Lam, S. Generalized Analytic Rule Extraction for Feedforward Neural Networks. *IEEE Transactions on Knowledge and Data Engineering* **1999,** *11,* 985–991.

[16] Haykin, S. *Neural Networks;* Pearson Education: Singapore, 2001.

[17] Ripley, B. *Pattern Recognition and Neural Networks;* Cambridge University Press: Oxford, 1996.

[18] Kohonen, T. *Self Organizing Maps;* volume 30 of *Springer Series in Information Sciences* Springer: Espoo, Finland, 1994.

[19] Hornik, K.; Stinchcombe, M.; White, H. Universal Approximation of an Unknown Mapping and its Derivation Using Multilayer Feedforward Networks. *Neural Networks* **1990,** *3,* 551–556.

[20] Hornik, K.; Stinchcombe, M.; White, H. Multilayer Feedforward Networks are Universal Approximators. *Neural Networks* **1989,** *2,* 35–366.

[21] Livingstone, D.; Manallack, D. Statistics Using Neural Networks: Chance Effects. *J. Med. Chem.* **1993,** *36,* 1295–1297.

[22] Wessel, M. *Computer Assisted Development of Quantitative Structure Property Relationships and Design of Feature Selection Routines,* Thesis, Department of Chemistry, Pennsylvania State University, 1997.

[23] Kalelkar, S.; Dow, E. R.; Grimes, J.; Clapham, M.; Hu, H. Automated Analysis of Proton NMR Spectra from Combinatorial Rapid Parallel Synthesis Using Self-Organizing Maps. *J. Comb. Chem.* **2002,** *4,* 622–629.

[24] Hoehn, F.; Lindner, E.; Mayer, H. A.; Hermle, T.; Rosenstiel, W. Neural Networks Evaluating NMR Data: An Approach To Visualize Similarities and Relationships of Sol-Gel Derived Inorganic-Organic and Organometallic Hybrid Polymers. *J. Chem. Inf. Comput. Sci.* **2002,** *42,* 36–45.

[25] Chen, L.; Gasteiger, J. Knowledge Discovery in Reaction Databases: Landscaping Organic Reactions by a Self Organizing Neural Network. *J. Am. Chem. Soc.* **1997,** *119,* 4033–4042.

[26] Bienfait, B. Applications of High Reolution Self Organizing Maps to Retrosynthetic and QSAR Analysis. *J. Chem. Inf. Comput. Sci.* **1994,** *34,* 890–898.

[27] Rose, V.; Croall, I.; Macfie, H. An Application of Unsupervised Neural Network Methodology Kohonen Topology-Preserving Mapping to QSAR Analysis. *Quant. Struct.-Act. Relat.* **1991,** *10,* 6–15.

[28] Anzali, S.; Barnickel, G.; Krug, M.; Sadowski, J.; Wagener, M.; Gasteiger, J.; Polanski, J. The Comparison of Geometric and Electronic Properties of Molecular Surfaces by Neural Networks: Application to the Analysis of Corticosteroid-Binding Globulin Activity of Steroids. *J. Comp. Aid. Molec. Des.* **1996,** *10,* 521–534.

[29] Chen, L.; Gasteiger, J. Knowledge Discovery in Reaction Databases: Landscaping Organic Reactions by a Self Organizing Neural Network. *J. Am. Chem. Soc.* **1997,** *119,* 4033–4042.

[30] Espinosa, G.; Arenas, A.; Giralt, F. An Integrated SOM Fuzzy ARTMAP Neural System for the Evaluation of Toxicity. *J. Chem. Inf. Comput. Sci.* **2002,** *42,* 343–359.

[31] Nicklaus, M. "`http://cactus.nci.nih.gov/DownLoad/AID2DA99.sdz`", 1999.

[32] Breiman, L. Statiscal Modeling: Two Cultures. *Statistical Science* **2001,** *16,* 199–231.

[33] Sutton, R.; Barto, A. *Reinforcement Learning;* MIT Press: Cambridge, 1999.

[34] Witten, W.; Frank, E. *Data Mining;* Morgan Kaufman: San Francisco, 2000.

[35] Christianini, N.; Shawn-Taylor, J. *An Introduction to Support Vector Machines;* Cambridge University Press: Cambridge, 2002.

[36] Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning;* Springer: New York, 2003.

[37] Kohonen, T. *Self-Organization and Associative Memory;* Springer-Verlag: Berlin, 1989.

[38] Breiman, L.; Friedman, J.; Olshen, R.; Stone, C. *Classification and Regresion Trees;* CRC Press: Boca Raton, FL, 1984.

[39] Zhang, X.; Singer, B. *Recursive Partitioning in the Health Sciences;* Springer: New York, 1999.

[40] Shaha, A. Implications Of Prognostic Factors And Risk Groups In The Management Of Differentiated Thyroid Cancer. *Laryngoscope* **2004,** *114,* 393–402.

[41] Schuurmann, G.; Aptula, A. O.; Kuhne, R.; Ebert, R. Stepwise Discrimination between Four Modes of Toxic Action of Phenols in the *Tetrahymena pyriformis* Assay. *Chem. Res. Tox.* **2003,** *16,* 974–987.

[42] Butina, D.; Gola, J. M. R. Modeling Aqueous Solubility. *J. Chem. Inf. Comput. Sci.* **2003,** *43,* 837–841.

[43] Bos, P.; Baars, B.; van Raaij, M. Risk Assessment Of Peak Exposure To Genotoxic Carcinogens: A Pragmatic Approach. *Tox. Lett.* **2004,** *151,* 43–50.

[44] Ebert, M. P. A.; Meuer, J.; Wiemer, J. C.; Schulz, H.-U.; Reymond, M. A.; Traugott, U.; Malfertheiner, P.; Rocken, C. Identification of Gastric Cancer Patients by Serum Protein Profiling. *J. Proteome Res.* **2004,** *3,* 1261–1266.

[45] Tong, W.; Xie, W.; Hong, H.; Fang, H.; Shi, L.; Perkins, R.; Petricoin, E. Using Decision Forest To Classify Prostate Cancer Samples On The Basis Of SELDI-TOF MS Data: Assessing Chance Correlation And Prediction Confidence. *Evironmental Health Perspectives* **2004,** *112,* 1622–1627.

[46] Therneau, T.; Atkinson, E. "An Introduction to Recursive Partitioning Using the RPART Routines", Technical Report, Department of Health Science Research, Mayo Clinic, Rochester, Minnesota, 1997.

[47] Venables, W.; Ripley, B. *Modern Applied Statistics with S;* Springer: New York, 2002.

[48] Breiman, L. Bagging Predictors. *Machine Learning* **1996,** *26,* 123–140.

[49] Breiman, L. Randomizing Outputs to Increase Prediction Accuracy. *Machine Learning* **2000,** *40,* 229–242.

[50] Dietterich, T. An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization. *Machine Learning* **2000,** *40,* 139–167.

[51] Svetnik, V.; Liaw, A.; Tong, C.; Culberson, C.; Sheridan, R.; Feuston, B. Random Forest: A Classification and Regression Tool for Compound Classification and QSAR Modeling. *J. Chem. Inf. Comput. Sci.* **2003,** *42,* 1947–1958.

[52] Breiman, L. "Out-of-bag estimation", Technical Report, Department of Statistics, University of California, Berkeley, 1996.

[53] Guha, R. Unpublished data.

[54] Cover, T.; Hart, P. Nearest Neighbor Pattern Classification. *Proc. IEEE Trans. Inform. Theory* **1967,** *IT-11,* 21–27.

[55] Friedman, J. "Flexible Metric Nearest Neighbor Classification", Technical Report, Stanford University, 1994.

[56] Hastie, T.; Tibshirani, R. Discriminant Adaptive Nearest Neighbor Classification. *IEEE Pattern Recognition and Machine Intelligence* **1996,** *18,* 607–616.

[57] Shen, M.; Xiao, Y.; Golbraikh, A.; Gombar, V.; Tropsha, A. Development and Validation of *k*-Nearest Neighbor QSPR Models of Metabolic Stability of Drug Candidates. *J. Med. Chem.* **2003,** *46,* 3013–3020.

[58] Gillet, V.; Khatib, W.; Willet, P.; Gleming, P.; Green, D. Combinatorial Library Design Using a Multiobjective Genetic Algorithm. *J. Chem. Inf. Comput. Sci.* **2002,** *42,* 375–385.

[59] Le Bailly de Tilleghem, C.; Beck, B.; Boulanger, B.; Govaerts, B. A Fast Exchange Algorithm for Designing Focused Libraries in Lead Optimization. *J. Chem. Inf. Comput. Sci.* **2005,** *ASAP,* XXX.

[60] Agrafiotis, D. K.; Rassokhin, D. N. Design and Prioritization of Plates for High-Throughput Screening. *J. Chem. Inf. Comput. Sci.* **2001,** *41,* 798–805.

[61] Altman, D. G.; Andersen, P. K. Bootstrap investigation of the stability of a Cox regression model. *Statistics in Medicine* **1989,** *8,* 771–783.

[62] Tibshirani, R. Regression Shrinkage and Selection via the Lasso. *J. Royal Stat. Soc. B* **1996,** *58,* 267–288.

[63] Mantel, N. Why Stepdown Procedures in Variable Selection. *Technometrics* **1970,** *12,* 621–625.

[64] Hinnela, J.; Saxen, H.; Pettersson, F. Modeling of the Blast Furnace Burden Distribution by Evolving Neural Networks. *Ind. Eng. Chem. Res.* **2003,** *42,* 2314–2323.

[65] Moisa, T.; Ontanu, D.; Dediu, A. *Speech Synthesis Using Neural Networks Trained by an Evolutionary Algorithm;* volume 2074 of *Lecture Notes in Computer Science* Springer-Verlag GmbH: New York, 2001.

[66] Mattioni, B. E.; Jurs, P. C. Prediction of Glass Transition Temperatures from Monomer and Repeat Unit Structure Using Computational Neural Networks. *J. Chem. Inf. Comput. Sci.* **2002,** *42,* 232–240.

[67] Guha, R.; Jurs, P. C. Development of Linear, Ensemble, and Nonlinear Models for the Prediction and Interpretation of the Biological Activity of a Set of PDGFR Inhibitors. *J. Chem. Inf. Comput. Sci.* **2004,** *44,* 2179–2189.

[68] Guha, R.; Jurs, P. The Development of QSAR Models To Predict and Interpret the Biological Activity of Artemisinin Analogues. *J. Chem. Inf. Comp. Sci.* **2004,** *44,* 1440–1449.

[69] Venkatraman, V.; Dalby, A. R.; Yang, Z. R. Evaluation of Mutual Information and Genetic Programming for Feature Selection in QSAR. *J. Chem. Inf. Comput. Sci.* **2004,** *44,* 1686–1692.

[70] Wright, T.; Gillet, V. J.; Green, D. V. S.; Pickett, S. D. Optimizing the Size and Configuration of Combinatorial Libraries. *J. Chem. Inf. Comput. Sci.* **2003,** *43,* 381–390.

[71] Goicoechea, H. C.; Olivieri, A. C. Wavelength Selection for Multivariate Calibration Using a Genetic Algorithm: A Novel Initialization Strategy. *J. Chem. Inf. Comput. Sci.* **2002,** *42,* 1146–1153.

[72] Hervás, C.; Silva, M.; Serrano, J. M.; Orejuela, E. Heuristic Extraction of Rules in Pruned Artificial Neural Networks Models Used for Quantifying Highly Overlapping Chromatographic Peaks. *J. Chem. Inf. Comput. Sci.* **2004,** *44,* 1576–1584.

[73] Leardi, R. Genetic Algorithms in Chemometrics and Chemistry. *J. Chemo.* **2001,** *15,* 559–569.

[74] Levine, B.; Moores, A. Genetic Algorithm in Analytical Chemistry. *Anal. Lett.* **1999,** *32,* 433–445.

[75] Metropolis, N.; Rosenbluth, A.; Rosenbluth, M.; Teller, A.; Teller, E. Equation of State Calculations by Fast Computing Machines. *J. Chem. Phys.* **1953,** *21,* 1087–1092.

[76] Bohachevsky, I.; Johnson, M.; Stein, M. Generalized Simulated Annealing for Function Optimization. *Technometrics* **1986,** *28*, 209–217.

[77] Sutter, J.; Kalivas, J. Comparison of Forward Selection, Backward Elimination and Generalized Simulated Annealing for Variable Selection. *Microchemical J.* **1993,** *47*, 60–66.