ELSEVIER

Computers and Chemistry 26 (2001) 5–14

Computers & Chemistry

www.elsevier.com/locate/compchem

# Drug design by machine learning: support vector machines for pharmaceutical data analysis

R. Burbidge *, M. Trotter, B. Buxton, S. Holden

*University College London, Gower Street, London WC1E 6BT, UK*

Received 20 November 2000; accepted 27 May 2001

**Abstract**

We show that the support vector machine (SVM) classification algorithm, a recent development from the machine learning community, proves its potential for structure–activity relationship analysis. In a benchmark test, the SVM is compared to several machine learning techniques currently used in the field. The classification task involves predicting the inhibition of dihydrofolate reductase by pyrimidines, using data obtained from the UCI machine learning repository. Three artificial neural networks, a radial basis function network, and a C5.0 decision tree are all outperformed by the SVM. The SVM is significantly better than all of these, bar a manually capacity-controlled neural network, which takes considerably longer to train. © 2001 Elsevier Science Ltd. All rights reserved.

*Keywords:* Machine learning; Support vector machines; Model selection; Structure–activity relationship analysis; Cheminformatics

## 1. Introduction

A recent classification algorithm from the machine learning community is introduced and applied to a well-known problem in the field of drug discovery. As a control, the algorithm is compared to several intelligent classification techniques that are currently used to tackle the problem. We first describe structure–activity relationship (SAR) analysis, a technique used by pharmaceuticals companies in the drug discovery process. Following this, we introduce the support vector machine (SVM), a new learning algorithm for classification and present empirical evidence for this approach in SAR analysis. Finally, we make some conclusions.

## 2. SAR analysis

The advent of combinatorial chemistry in the mid-1980s has allowed the synthesis of hundreds, thousands and even millions of new molecular compounds at a time. Nevertheless, even this level of compound production will fall short of exhausting the trillions of potential combinations within a few thousand years. The need for a more refined search than simply producing and testing every single molecular combination possible has meant that statistical methods and, more recently, intelligent computation have become an integral part of the drug production process. Structure–activity relationship (SAR) analysis is one technique used to reduce the search for new drugs. It also presents an extremely challenging problem to the field of intelligent systems. A successful solution to this problem has the potential to provide significant economic benefit via increased process efficiency. We present evidence that a recently developed, state-of-the-art machine learning technique outperforms several of its competitors when applied to such a problem.

The underlying assumption behind SAR analysis is that there is a relationship between the variation of biological activity within a group of molecular compounds with the variation of their respective structural

* Corresponding author.


PII: S0097-8485(01)00094-8

and chemical features. The analyst searches for a rule or function that predicts a molecule's activity from the values of its physicochemical descriptors. The aim of SAR analysis is to discover such general rules and equations.

It is common to prefix the acronym SAR with either 'Q' or 'q', indicating 'quantitative' or 'qualitative', respectively. QSAR involves modelling a continuous activity for quantitative prediction of the activity of previously unseen compounds. qSAR aims to separate the compounds into a number of discrete class types, such as 'active' and 'inactive' or 'good' and 'bad'. Confusion can arise due to differing use of the two terms. For example, discrete classification problems are sometimes referred to as QSAR, as they quantify the activity types of the compounds examined. A good rule of thumb is that QSAR analysis is a regression problem and qSAR analysis a classification problem.

Activities of interest include chemical reactivity, biological activity and toxicity. In this experiment, we wish to predict qualitative biological activity for drug design, using a publicly available data set. Typically, the analyst examines the descriptors of a finite number of compounds in relation to a known target activity and attempts to model the relationship between them. A priori knowledge of the underlying chemistry may also be considered. The methods used to represent molecular features when producing a SAR vary greatly and a lot of work is carried out on this sub-problem alone. The problem of choosing a suitable representation will not feature further in this paper, however, as we use that given by those who produced the data. The aim of SAR analysis is to discover rules that successfully predict the activities of previously unseen compounds. Solving SAR problems where data sets contain few compounds and many descriptors is difficult with standard statistical approaches. Data sets which describe highly non-linear relationships between structure and activity, pose similar problems.

SAR analysis is applied to many areas of modern drug design and production, on both the small, medium and large scales. Small-scale problems, such as the one presented here, are commonly found when designing combinatorial libraries in the early stages of the production process. Due to the number of potential molecular combinations from existing molecular databases, the search for potentially useful combinations must be refined. This can be done by discriminating between molecules that are likely to produce the desired effect upon synthesis and those that are not—a process termed 'virtual' or 'in silico' screening. In doing so, the drug designer can build up small sub-libraries of suitable molecules taken from a larger and more diverse library, greatly reducing the search in doing so. SAR analysis is used to find suitable molecules for the sub-library. Accuracy is important in the creation of combinatorial libraries, as to miss an interesting molecule is lose a potential lead for drug discovery.

Artificial intelligence techniques have been applied to SAR analysis since the late 1980s, mainly in response to increased accuracy demands. Intelligent classification techniques that have been applied to this problem include neural networks (Devillers, 1999b), genetic algorithms (Devillers, 1999a) and decision trees (Hawkins et al., 1997). The problem of combining high classification accuracy with informative results has also been tackled via the use of hybrid and prior knowledge techniques, such as expert systems (Gini et al., 1998). Machine learning techniques have, in general, offered greater accuracy than have their statistical forebears, but there exist accompanying problems for the SAR analyst to consider. Neural networks, for example, offer high accuracy in most cases but can suffer from overfitting the training data (Manallack and Livingstone, 1999). Other problems with the use of neural networks concern the reproducibility of results, due largely to random initialization of the network and variation of stopping criteria, and lack of information regarding the classification produced (Manallack and Livingstone, 1999). Genetic algorithms can suffer in a similar manner. The stochastic nature of both population initialization and the genetic operators used during training can make results hard to reproduce (Goldberg, 1989). Decision trees offer a large amount of information regarding their decisions, in the form of predictive rules. There exist powerful decision tree methods, such as OC1 (Murthy et al., 1994), capable of competing with the previous two techniques, but they commonly require a great deal of tuning to do so. Simpler techniques are available and popular but, as displayed in our results, can fail to attain similar performance levels.

Owing to the reasons outlined above, there is a continuing need for the application of more accurate and informative classification techniques to SAR analysis.

## 3. Problem description

The data used in this experiment were obtained from the UCI Data Repository (Blake and Merz, 1998) and are described in King et al. (1992). The problem is to predict the inhibition of dihydrofolate reductase by pyrimidines. The biological activity is measured as $\log(1/K_i)$, where $K_i$ is the equilibrium constant for the association of the drug to dihydrofolate reductase. QSAR problems are generally formulated as regression problems, i.e. learn the posterior probability of the target, $y$, given some predictive attributes, $\mathbf{x}$. In this case the target is $\log(1/K_i)$ and the attributes are as follows. Each drug has three positions of possible substitution. For each substitution position there are nine

descriptors: polarity, size, flexibility, hydrogen-bond donor, hydrogen-bond acceptor, π donor, π acceptor, polarizability and σ effect. Each of the 24 non-hydrogen substituents in the compound set was given an integer value for each of these properties (see King et al. (1992) for details); lack of a substitution is indicated by nine $-1$s. This gives 27 integer attributes for each drug. The approach here is to avoid solving a regression problem by recasting it as one suitable for classification or inductive logic programming (ILP), the ILP approach is described in King et al. (1992). Here we focus on the classification problem.

The task is to learn the relationship great($d_n$, $d_m$) which states that drug no. $n$ has a higher activity than drug no. $m$. Each instance consists of two drugs, giving 54 attributes in total, and a label 'true' or 'false', indicating the value of the relationship great(). There are 55 compounds in the data set. In order to obtain a good estimate of the performance of the algorithms, the data were partitioned into a five-fold cross-validation series as follows. The 55 compounds were randomly partitioned into 5 sets of 11 compounds. Each fold consisted of a training set $A$ of 44 compounds and a test set $B$ of 11 compounds. The classification training data were constructed by labelling each pair $(n, m) \in A \times A$ according to the true value of great($d_n$, $d_m$). This gives $44 \times 43 = 1892$ examples of great(), in practice, however, the training sets were slightly smaller since great() is not defined when the two drugs have the same activity. The classification test data were constructed by forming all pairs $(n, m) \in (A \times B) \cup (B \times A) \cup (B \times B)$ (i.e. all comparisons not in the classification training set) which gives slightly less than $44 \times 11 + 11 \times 44 + 11 \times 10 = 1078$ examples of great().

Due to the above construction each instance in the data is followed by its inverse, for example if the first instance represents great($d_2$, $d_1$) = true then the second instance represents great($d_1$, $d_2$) = false. This produces a symmetric two-class classification problem where the classes are equal in size and have equal misclassification costs.

Each of the training sets was then used to train an algorithm to produce a classification rule. This classification rule can then be used to predict which of two unseen compounds has the greatest activity. The generalization ability of such a rule, that is, the probability that it will correctly rank two unseen compounds in terms of activity, is estimated as follows. Each of the trained classifiers is used to classify the corresponding test set. Each of these five error rates is an estimate of the true error rate, the final estimate of error rate is taken to be their mean. Note that even if the sets of training and test compounds are drawn independently, by construction, the classification test sets are not independent of the classification training sets. Thus, this estimate of true error rate will be biased downwards and will give an optimistic estimate of the error rate on unseen cases. Nevertheless, it is adequate for a *comparison* of techniques. In the following sections 'training set' will mean the set of approximately 1800 labelled instances of great() and similarly for 'test set'.

Finally, note that in learning the relationship great() we can rank the compounds in terms of their activity without going to the effort of predicting that activity. This is in contrast to the more general and harder problem of learning a set of rules or a regression equation to predict activity. Formulated as a classification problem there are approximately $55 \times 54 = 2970$ examples in 54 dimensions. This is still a small problem in comparison to those that could be formulated by the pharmaceuticals industry. We have chosen this problem to demonstrate the utility of the SVM in cheminformatics, see Section 6 below for a discussion.

## 4. Learning algorithms

Classifiers typically learn by empirical risk minimization (ERM) (Vapnik, 1998), that is they search for the hypothesis with the lowest error on the training set. Unfortunately, this approach is doomed to failure without some sort of capacity control (Bishop, 1995; Burges, 1998). To see this, consider a very expressive hypothesis space. If the data are noisy, which is true of most real world applications, then the ERM learner will choose a hypothesis that accurately models the data and the noise. Such a hypothesis will perform badly on unseen data. To overcome this we limit the expressiveness of the hypothesis space. In the remainder of this section are described the learning algorithms used and the methods of capacity control. Note that the SVM is the only algorithm which performs capacity control simultaneously with risk minimization, this is termed structural risk minimization (SRM).

In the following the Euclidean norm of a $d$-dimensional vector $\mathbf{x} = (x_1, \ldots, x_d)$ is denoted by $\|\cdot\|$, and is defined to be

$$\|\mathbf{x}\|^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{d} x_i^2,$$

where $\mathbf{x}$ is a column vector and superscript T denotes the transpose.

### 4.1. Neural networks

Neural networks are a biologically inspired form of distributed computation. There are a large number of interconnected nodes that perform summation and thresholding, in loose analogy with the neurons of the brain. Feed-forward neural networks can be used to learn a real-valued or a discrete-valued decision rule; the latter case is, from a statistical learning theory

standpoint, essentially an extension of linear discriminant classification (Duda and Hart, 1973). Neural networks do not explicitly model the underlying distribution but in certain cases the outputs can be treated in a rigorous probabilistic manner.

Neural networks have been successfully applied in many settings, including speech recognition (Lang et al., 1990), handwritten digit recognition (LeCun et al., 1989) and driving a car (Pomerleau, 1993). Nevertheless, they suffer from many problems and are not well-controlled learning machines (Vapnik, 1998). Neural networks have been proposed as a useful tool for many areas of drug design (Schneider, 2000), and have exhibited good performance in optimizing chemical libraries (Sadowski, 2000) and SAR analysis (Kövesdi et al. 1999). However, in many cases the reasoning behind choosing a particular architecture or training algorithm is unclear, and model selection and training times are rarely reported. This is not too important in small-scale drug design problems, but will become increasingly significant with the growth of pharmaceutical data sets. In this section we introduce the perceptron and some extensions, the multi-layer perceptron and the radial basis function network.

#### 4.1.1. Multi-layer perceptrons

The perceptron (Rosenblatt, 1958) is an information processing unit of the form

$$f_j(\mathbf{x}, \mathbf{w}) = \mathrm{sgn}\left(\sum_i w_{ji} x_i - \theta_j\right) \quad (1)$$

that is, the output of a perceptron at node $j$, $f_j$, is the weighted sum of its inputs, $x_i$, thresholded at $\theta_j$. Such a perceptron is able to represent all decision hyperplanes over the space of its inputs $\mathbf{x} \in X$. In order to learn more complex decision functions the inputs $x_i$ are fed into a number of perceptrons nodes, each with its own set of weights and threshold. The outputs of these nodes are then input into another layer of nodes and so on, the output of the final layer of nodes is the output of the network. Such a network is termed a *multi-layer perceptron* (MLP) and the layers of nodes whose input and output are seen only by other nodes are termed *hidden*. An MLP with enough hidden layers and nodes can approximate any decision function to an arbitrary degree of accuracy (Bishop, 1995), but this result is not much use in practice as it does not specify the architecture or the values of the weights and thresholds.

Normally the architecture of the MLP is specified in advance and the weights and biases are estimated by supervised learning. In order to propagate errors through the network the sgn function in Eq. (1) is replaced by a differentiable sigmoid transfer function such as tanh. See Hertz et al. (1991), Bishop (1995) for a discussion of supervised training algorithms for MLPs.

MLPs suffer from many problems, training can take many iterations to converge and the speed is dependent on a number of parameters which specify the learning rate, a momentum term and stopping criterion. Also, MLPs are prone to over-fitting without some sort of capacity control. There are three methods of capacity control for a MLP: early stopping, network complexity and weight decay. The method of early stopping tracks the network's performance using a separate *validation* set. This is usually a subset of the training data that has been held out. Typically the error on the validation set will decrease as the network fits the data, and then increase as the network fits the idiosyncrasies of the noise in the training data. The error rate on the validation set is taken as an estimate of the true error rate. The second method is to train a series of networks with an increasing number of hidden nodes. Increasing the number of hidden nodes increases the expressiveness of the hypothesis space of the network. Using a validation set the number of hidden nodes can be chosen to minimize expected error. The third option is to penalize overly complex networks during training, e.g. by adding the norm of the weight vector to the error function. This amounts to decaying the weights during training. Forcing weights to take small values reduces the expressiveness of the hypothesis space (Bishop, 1995). This requires setting further parameters for the weight decay which can affect the quality of the solution. Another methods of avoiding over-fitting in neural networks is to add noise (termed 'jitter') to the inputs. All of these methods suffer from requiring a number of parameters to be set that affect the rate of convergence and the quality of the solution. Model combination and Bayesian methods can partially overcome these methods, but require many models to be trained and are hence computationally expensive.

#### 4.1.2. Implementation

The MLPs used in this experiment were implemented on a 400 MHz NT workstation with 384 Mb RAM using the software package CLEMENTINE 5.1 from SPSS (SPSS, 1999).

CLEMENTINE trains neural networks by gradient descent. A momentum term is added to the weight update to avoid local minima. It is necessary to specify a variety of parameters in CLEMENTINE that can affect the rate of convergence and the quality of the solution. Most of these were set at the defaults estimated automatically by CLEMENTINE on the basis of the size of the data. The momentum term $\alpha$ was set at 0.9, the learning rate $\eta$ is initially 0.3, decays exponentially to 0.01, is then reset to 0.1, and then decays to 0.01 in 30 epochs (an epoch is one complete pass through the training data). Decaying the learning rate ensures that convergence is initially rapid and that the algorithm does not 'over-shoot' a good solution. The algorithm was halted

when there had been no significant reduction in training error for 100 epochs.

The only remaining parameters concerned the architecture of the network. Three methods were used to optimize the architecture. The first method was to train a sequence of two-layer neural networks with an increasing number of hidden nodes. As the number of hidden nodes increases, the training error and true error decrease. However, increasing the number of hidden nodes past a certain (data dependent) value, while further decreasing training error, will increase true error: an effect known as over-fitting. In order to prevent over-fitting 20% of the training data were held out as a validation set. The number of hidden nodes was varied from 2 to 54 in steps of 2. The optimum number of hidden nodes (20) was taken to be that which minimized the error rate on the validation set. The number of hidden nodes was chosen from only the first cross-validation fold. It would be preferable to repeat this model selection procedure for each of the five folds, but this takes a prohibitively long time, see Section 5 below. The second method was to dynamically grow a network until no further improvement is achieved. The drawback of this method is that it is computationally expensive. There are no parameters for this method in CLEMENTINE. The third method was to start from a large network and prune hidden nodes and input nodes to obtain as compact a network as possible, such a network is expected to have low generalization error. This method is also computationally expensive. There are numerous parameters in CLEMENTINE for this method which were set at the defaults due to the extensive training time. For these last two methods 20% of the available training data were used to estimate generalization error in choosing the optimal architecture. The networks were then retrained with the optimal architecture using all of the available training data.

#### 4.1.3. Radial basis function networks

A Gaussian radial basis function classifier is a function

$$f(\mathbf{x}) = \operatorname{sgn}\left[\sum_{i=1}^{m} w_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}_i\|^2}{2\sigma_i^2}\right) + b\right]$$

where $b$, $\sigma_i$ are constants. That is, a new point $\mathbf{x}$ is compared to a collection of prototypes $\mathbf{c}_i$ $i = 1,\ldots,m$, the similarity measures, in this case Gaussian, are weighted and summed. If this quantity is not less than $b$ then $\mathbf{x}$ is assigned to class 'true', otherwise to class 'false'. Such a weighted sum of Gaussian bumps can approximate any decision boundary arbitrarily well, given enough basis centres $m$ and appropriate values for the weights $w_i$ and the bias $b$. Given a collection of centres the weights and bias can be computed by gradient descent as for an MLP (Bishop, 1995). The RBF centres $\mathbf{c}_i$ and widths $\sigma_i$ must be chosen in advance, typically this is done by $k$-means clustering. Training an RBF network is a two-stage process, in contrast to training an MLP, for which all weights are optimized simultaneously. The quality of an RBF solution thus depends on the quality of the clustering, which depends on the clustering algorithm and the choice of $k$.

#### 4.1.4. Implementation

The RBF networks used in this experiment were implemented on a 400 MHz NT workstation with 384 Mb RAM using the software package CLEMENTINE 5.1 from SPSS (SPSS, 1999).

As for the MLP, CLEMENTINE has a number of tunable parameters which affect the speed and accuracy of the algorithm and the final classifier. The momentum term $\alpha$ was set at 0.9, the learning rate $\eta$ was computed automatically by CLEMENTINE. The stopping criterion was as for the MLP. Capacity control was performed as for the MLP, by varying the number of RBF centres, $k$, and using 20% of the data to estimate the optimal number (14). CLEMENTINE uses $k$-means clustering to determine the RBF centres $\mathbf{x}_i$ and widths $\sigma_i$.

### 4.2. Decision trees

Decision tree learning can provide an informative model, through which predictive rules are induced to solve classification problems (Breiman et al., 1984). The method uses a process called recursive partitioning. In their simplest form, e.g. C4.5 (Quinlan, 1992), each attribute of the data is examined in turned and ranked according to its ability to partition the remaining data. The data are propagated along the branches of the tree until sufficient attributes have been chosen to correctly classify them. The trained classifier has a tree-like structure. Each 'leaf' of the tree represents a subset of the data that lies wholly in one class. Decision trees have a tendency to over-fit the training data. To allow for noise in the data the tree can be pruned to allow for misclassifications in the training data. Pruning is a heuristic process, usually requiring the use of a hold-out validation set to determine the optimal tree structure (Breiman et al., 1984). There may exist several tree structures (hypotheses) capable of achieving a given training accuracy. Decision tree learning algorithms are biased to select the simplest of these trees. Decision trees can also be constructed where the branching criteria are allowed to be linear or non-linear combinations of attributes, instead of just splitting on a single attribute. These methods, e.g. OC1 (Murthy et al., 1994), are more powerful but require extensive model selection. These methods produce shorter trees, with more complex branching rules.

#### 4.2.1. Implementation

The decision tree used in this experiment was the implementation of C5.0 in the software package CLEMENTINE 5.1 (SPSS, 1999). This algorithm is a modification of the well-known C4.5 algorithm described in Quinlan (1992). This was implemented on a 400 MHz NT workstation with 384 Mb of RAM.

The default parameter settings were used, as these gave generalization performance very close to that achieved by model selection. The pruning severity was set at 75% and the algorithm was used to learn a rule set as opposed to a tree, as this provided slightly higher generalization accuracy.

### 4.3. Support vector machines

The support vector machine was developed by Vapnik (1979) as an implementaton of SRM. The idea behind SRM is given a sequence of hypothesis spaces of increasing complexity, choose from each that which minimizes training error. Then from this sequence of hypotheses choose that which minimizes an upper bound on the generalization error. The SVM approximates performing these two steps simultaneously. The SVM achieves this by controlling the size of the feature weights. This is a principled formulation of weight decay used in neural networks to improve generalization (Bishop, 1995; Hertz et al., 1991). The geometrical interpretation is that an SVM chooses the optimal separating surface, i.e. the hyperplane that is, in a sense, equidistant from the two classes (Burges, 1998). This optimal separating hyperplane has many nice statistical properties as detailed in Vapnik (1998). The SVM method is outlined first for the linearly separable case. Kernel functions are then introduced in order to deal with non-linear decision surfaces. Finally, for noisy data, when complete separation of the two classes may not be desirable, slack variables are introduced. See Osuna et al. (1997), Burges (1998) for tutorials on SVMs.

#### 4.3.1. Linear decision surfaces

Denote a true example of great() by $\mathbf{x}\in T$ and a false example as $\mathbf{x}\in F$. If the training data are linearly separable then there exists a pair $(\mathbf{w}, b)$ such that

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}+b\geq +1, \quad \text{for all } \mathbf{x}\in T \tag{2}$$

$$\mathbf{w}^{\mathrm{T}}\mathbf{x}+b\leq -1, \quad \text{for all } \mathbf{x}\in F \tag{3}$$

with the decision rule given by

$$f_{\mathbf{w},b}(\mathbf{x})=\mathrm{sgn}(\mathbf{w}^{\mathrm{T}}\mathbf{x}+b), \tag{4}$$

$\mathbf{w}$ is termed the weight vector and $b$ the bias (or $-b$ is termed the threshold). The inequality constraints (2) and (3) can be combined to give

$$y(\mathbf{w}^{\mathrm{T}}\mathbf{x}+b)\geq 1, \quad \text{for all } \mathbf{x}\in T\cup F. \tag{5}$$

Without loss of generality the pair $(\mathbf{w}, b)$ can be rescaled such that

$$\min_{i=1,\ldots,l}\left|\mathbf{w}^{\mathrm{T}}\mathbf{x}_i+b\right|=1,$$

this constraint defines the set of canonical hyperplanes on $\Re^d$. In order to restrict the expressiveness of the hypothesis space, the SVM searches for the simplest solution that classifies the data correctly. The learning problem is hence reformulated as: minimize $\|\mathbf{w}\|^2$ subject to the constraints of linear separability (5). This is equivalent to maximizing the distance, normal to the hyperplane, between the convex hulls of the two classes; this distance is called the margin. The optimization is now a quadratic programming (QP) problem

$$\underset{\mathbf{w},b}{\text{Minimize}}\ \Phi(\mathbf{w})=\frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{subject to } y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i+b)\geq 1, \quad i=1,\ldots,l$$

This problem has a global optimum, thus the problems of many local optima in the case of an MLP is avoided. This has the advantage that parameters in a QP solver will affect only the training time, and not the quality of the solution. This problem is tractable but in order to proceed to the non-separable and non-linear cases it is useful to consider the dual problem as outlined below. Full details of this and of the relevant results from optimization theory are given in Vapnik (1998). The Lagrangian for this problem is

$$L(\mathbf{w}, b, \Lambda)=\frac{1}{2}\|\mathbf{w}\|^2-\sum_{i=1}^{l}\lambda_i[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i+b)-1], \tag{6}$$

where $\Lambda=\{\lambda_1,\ldots,\lambda_l\}$ are the Lagrange multipliers, one for each data point. The solution to this quadratic programming problem is given by maximizing $L$ with respect to $\Lambda\geq 0$ and minimizing with respect to $\mathbf{w}$, $b$. Differentiating with respect to $\mathbf{w}$ and $b$ and setting the derivatives equal to 0 yields

$$\frac{\partial L(\mathbf{w}, b, \Lambda)}{\partial \mathbf{w}}=\mathbf{w}-\sum_{i=1}^{l}\lambda_i y_i\mathbf{x}_i=0$$

and

$$\frac{\partial L(\mathbf{w}, b, \Lambda)}{\partial b}=\sum_{i=1}^{l}\lambda_i y_i=0. \tag{7}$$

So that the optimal solution is given by

$$\mathbf{w}^*=\sum_{i=1}^{l}\lambda_i^*\mathbf{x}_i \tag{8}$$

substituting Eqs. (7) and (8) into Eq. (6) we can write

$$F(\Lambda)=\sum_{i=1}^{l}\lambda_i-\frac{1}{2}\|\mathbf{w}\|^2=\sum_{i=1}^{l}\lambda_i-\frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}\lambda_i\lambda_j y_i y_j\mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j, \tag{9}$$

which, written in matrix notation, leads to the following dual problem:

$$\text{Maximize } F(\Lambda) = \Lambda^{\mathrm{T}}1 - \frac{1}{2}\Lambda^{\mathrm{T}}D\Lambda \tag{10}$$

$$\text{subject to } \Lambda^{\mathrm{T}}Y = 0, \quad \Lambda \geq 0 \tag{11}$$

where $Y = (y_1,\ldots,y_l)$ and $D$ is a symmetric $l \times l$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i^{\mathrm{T}}\mathbf{x}_j$. Note that the Lagrange multipliers are only non-zero when $y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) = 1$, vectors for which this is the case are called *support vectors* since they lie closest to the separating hyperplane. The optimal weights are given by Eq. (8) and the bias is given by

$$b^* = y_i - \mathbf{w}^{*\mathrm{T}}x_i \tag{12}$$

for any support vector $\mathbf{x}_i$ (although in practice it is safer to average over all support vectors (Burges, 1998)). The decision function is then given by

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i\lambda_i^*(\mathbf{x}^{\mathrm{T}}\mathbf{x}_i) + b^*\right). \tag{13}$$

The solution obtained is often sparse since only those $\mathbf{x}_i$ with non-zero Lagrange multipliers appear in the solution. This is important when the data to be classified are very large, as is often the case in cheminformatics.

*4.3.2. Non-linear decision surfaces*

A linear classifier may not be the most suitable hypothesis for the two classes. The SVM can be used to learn non-linear decision functions by first mapping the data to some higher dimensional *feature space* and constructing a separating hyperplane in this space. Denoting the mapping to feature space by

$$X \to \mathscr{H},$$

$$\mathbf{x} \mapsto \phi(\mathbf{x}),$$

the decision functions (4) and (13) become

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}(\phi(\mathbf{x})^{\mathrm{T}}\mathbf{w}^* + b^*) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{l} y_i\lambda_i^*\phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{x}_i) + b^*\right). \end{aligned} \tag{14}$$

Note that the input data appear in the training (9) and decision functions (13) only in the form of inner products $\mathbf{x}^{\mathrm{T}}\mathbf{z}$, and in the decision function (14) only in the form of inner products $\phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{z})$. Mapping the data to $\mathscr{H}$ is time consuming and storing it may be impossible, e.g. if $\mathscr{H}$ is infinite dimensional. Since the data only appear in inner products, we require a computable function that gives the value of the inner product in $\mathscr{H}$ without explicitly performing the mapping. Hence, introduce a *kernel function*,

$$K(\mathbf{x}, \mathbf{y}) \equiv \phi(\mathbf{x})^{\mathrm{T}}\phi(\mathbf{z}),$$

so that the decision function (14) becomes

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{l} y_i\lambda_i^*K(\mathbf{x}, \mathbf{x}_i) + b^*\right)$$

and training is the same as Eq. (10) with $D_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$.

The only remaining problem is specification of the kernel function, as indicated in Osuna et al. (1997), the kernel should be easy to compute, well defined and span a sufficiently rich hypothesis space. Osuna et al. (1997) suggest a number of kernels derived from various mappings, including finite-, countable- and uncountable-dimensional feature spaces. A more common approach is to define a positive definite kernel that corresponds to a known classifier such as an RBF, two-layer MLP or polynomial classifier. This is possible since the theorem of Mercer (1909) states that any positive definite kernel corresponds to an inner product in some feature space.[1] Kernels can also be constructing to incorporate domain knowledge (Brown et al., 1997).

*4.3.3. Soft margin hyperplanes*

So far we have restricted ourselves to the case where the two classes are noise free. In the case of noisy data, forcing zero training error will lead to poor generalization. To take account of the fact that some data points may be misclassified, introduce a set of slack variables $\Xi = \{\xi_i\}_{i=1}^{l}$ that measure the amount of violation of the constraints (5). The problem can then be written

$$\begin{aligned} &\underset{\mathbf{w},b,\Xi}{\text{Minimize}}\ \Phi(\mathbf{w}, \Xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{l}\xi_i \\ &\text{subject to } y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1,\ldots,l, \end{aligned}$$

where $C$ is specified beforehand. $C$ is a regularization parameter that controls the trade-off between maximizing the margin and minimizing the training error. If $C$ is too small then insufficient stress will be placed on fitting the training data. If $C$ is too large then the algorithm will overfit the training data. Due to the statistical properties of the optimal separating hyperplane, $C$ can be chosen without the need for a hold-out validation set, as described in Section 4.3.4. The Lagrangian for this problem is

$$\begin{aligned} &L(\mathbf{w},b,\Lambda,\Xi,\Gamma) \\ &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l}\lambda_i[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{l}\gamma_i\xi_i \\ &\quad + C\sum_{i=1}^{l}\xi_i, \end{aligned} \tag{15}$$

where $\Lambda = \{\lambda_1,\ldots,\lambda_l\}$,, as before, and $\Gamma = \{\gamma_1,\ldots,\gamma_l\}$ are the Lagrange multipliers corresponding to the positivity of the slack variables. The solution of this problem is the saddle point of the Lagrangian given by minimizing $L$ with respect to $\mathbf{w}$, $\Xi$ and $b$, and maximizing with

[1] Specifically, this is a Hilbert space, i.e. it is a complete inner product space, this is required to maintain the statistical properties.

respect to $\Lambda \geq 0$ and $\Gamma \geq 0$. Differentiating with respect to $\mathbf{w}$, $b$ and $\Xi$ and setting the results equal to zero we obtain

$$\frac{\partial L(\mathbf{w}, b, \Lambda, \Xi, \Gamma)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} \lambda_i y_i x_i = 0,$$
$$\frac{\partial L(\mathbf{w}, b, \Lambda, \Xi, \Gamma)}{\partial b} = \sum_{i=1}^{l} \lambda_i y_i = 0 \quad (16)$$

and

$$\frac{\partial L(\mathbf{w}, b, \Lambda, \Xi, \Gamma)}{\partial \Xi} = C - \lambda_i - \gamma_i = 0. \quad (17)$$

So that the optimal weights are again given by

$$\mathbf{w}^* = \sum_{i=1}^{l} \lambda_i^* y_i \mathbf{x}_i. \quad (18)$$

Substituting Eqs. (16)–(18) into Eq. (15) gives the following dual problem

$$\text{Maximize } F(\Lambda) = \Lambda^{\mathrm{T}} I - \frac{1}{2}\Lambda^{\mathrm{T}} D \Lambda$$
$$\text{subject to } \Lambda^{\mathrm{T}} Y = 0, \quad 0 \leq \Lambda \leq C1,$$

where $Y = (y_1, \ldots, y_l)$ and $D$ is a symmetric $l \times l$ matrix with elements $D_{ij} = y_i y_j \mathbf{x}_i^{\mathrm{T}} \mathbf{x}_j$. The decision function implemented is exactly as before in Eq. (13). The bias term $b^*$ is given by Eq. (12) where $\mathbf{x}_i$ is a support vector for which $0 < \lambda_i < C$. There is no proof that such a vector exists but empirically this is usually the case. The soft margin formulation can be extended to incorporate kernels exactly as in Section 4.3.2.

#### 4.3.4. Implementation

The SVMs used in this experiment were trained using the SVM$^{\text{light}}$ pacakage described in Joachims (1999), on a SunOS Ultra-2 with two processors. Thus, a strict time comparison is not possible; recently SVM$^{\text{light}}$ has been ported to NT, and initial experiments suggest similar training times on these data.

The kernel function used was a Gaussian RBF

$$K(\mathbf{x}, \mathbf{z}) = \exp\left( -\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2} \right),$$

this projects the data into an countably infinite dimensional feature space. The data are separable in this space for a suitable value of $\sigma$, however to avoid overfitting the soft margin formulation of the SVM is used. This leaves two free parameters to be specified, $\sigma$ and $C$. These were selected by training an SVM for various values of these parameters, and selecting that pair which minimized an upper bound on the expected generalization error. This bound is calculated by SVM$^{\text{light}}$, as described in Joachims (2000). The values tried were $\sigma \in \{\sigma_0, 2\sigma_0, 3\sigma_0\}$ and $C \in \{C_0, 10C_0, 100C_0\}$, where $\sigma_0$ is the median separation of all $\mathbf{x} \in T$ to their nearest neighbour in $F$ (Brown et al., 1997); and, $C_0 = R^{-2}$ where $R$ is the radius of the smallest ball in feature space containing all of the data, $R$ can be approximated by 1 for an RBF kernel (Cristianini and Shawe-Taylor, 2000). Note that model selection is much easier for an SVM than for neural network algorithms, there are no learning rates or decay terms, and the free parameters can be set without use of a hold-out validation set.

Table 1
Test accuracies and training times averaged over the five cross-validation folds

| Algorithm | Error | $p$ | Time (s) |
|---|---|---|---|
| SVM-RBF | 0.1269 | 0.5000 | 93 |
| MLP | 0.1381 | 0.1894 | 857 |
| RBF network | 0.2272 | 0.0000 | 199 |
| 'Prune' network | 0.1620 | 0.0070 | 692 |
| 'Dynamic' network | 0.1488 | 0.0067 | 613 |
| C5.0 | 0.1870 | 0.0000 | 4 |

Also shown are $p$ values for the null hypothesis that the SVM-RBF is not peforming better than the other algorithms. The SVM outperforms all of the algorithms except the MLP, at the 99% significance level. Training and model selction for the SVM are an order of magnitude less than for MLP (see Table 2).

## 5. Results and discussion

The test errors averaged over the five cross-validation folds are shown in Table 1. Also shown are the training times, averaged over the cross-validation folds. If speed is the key issue then the C5.0 decision tree is the preferable algorithm; this, though has poor generalization accuracy in comparison to the SVM and sigmoid neural networks. The RBF algorithm fared poorly on this data, which may due to the quality of the clustering. The neural network with manual capacity control did not perform significantly worse than the SVM.

The key point here however is the problem of model selection. Four different neural network architectures were used on these data, all of them requiring heuristic methods for model selection which are computationally intensive. The total times for model selection and validation are shown in Table 2. Note that once a neural network architecture has been selected on the basis of the hold-out set it is then retrained using all of the training data. In contrast, once the SVM parameters have been selected, no further training is necessary, as all of the available data have already been used. Hence, the times in Table 2 include the time in Table 1 for the SVM, but not for the neural networks. Again if speed is the most important issue then C5.0 would be chosen as model selection provided no increase in accuracy. This is not a general rule however as decision trees usually

Table 2
Model selection time for the SVM and the neural networks

| Algorithm | Time (s) |
|---|---|
| SVM-RBF | 1800 |
| MLP | 20 715 |
| RBF network | 11 719 |
| 'Prune' network | 13 338 |
| 'Dynamic' network | 4748 |

The time for the SVM includes the training time in Table 1 since it is not necessary to retrain an SVM once the model has been selected. No model selection was performed for C5.0.

require extensive model selection to avoid overfitting (Breiman et al., 1984).

## 6. Conclusions

The support vector machine has been introduced as a robust and highly accurate intelligent classification technique, likely to be well suited to SAR analysis. On a simple but real SAR analysis problem the SVM outperforms several of the most frequently used machine learning techniques. An MLP achieves similar performance to that of the SVM but requires an order of magnitude longer training times and manual capacity control which is computationally expensive. This becomes increasingly significant when learning SARs on large numbers of compounds. Other techniques, including an RBF network, and a C5.0 decision tree, all fall considerably short of the SVM's performance. The SVM is an automated and efficient deterministic learning algorithm, providing the benefit of reproducible and verifiable results. It has the advantage over the other techniques of converging to the global optimum, and not to a local optimum that depends on the initialization and parameters affecting rate of convergence.

The preliminary evidence presented in this paper suggests that the SVM is a data-mining tool with great potential for QSAR analysis.

## Acknowledgements

This research has been undertaken within the INTErSECT Faraday Partnership managed by Sira Ltd and the National Physical Laboratory, and has been supported by the Engineering and Physical Sciences Research Council (EPSRC), GlaxoSmithKline, SPSS and Sira. The data mining software package CLEMENTINE has been provided by SPSS. R.B. is an associate of the Postgraduate Training Partnership established between Sira Ltd and University College London. Postgraduate Training Partnerships are a joint initiative of the Department of Trade and Industry and EPSRC.

## References

Bishop, C., 1995. Neural Networks for Pattern Recognition. Clarendon Press, Oxford.

Blake, C.L., Merz, C.J., 1998. UCI Repository of Machine Learning Databases, URL: http://www.ics.uci.edu/~mlearn/MLRepository.html.

Breiman, L., Friedman, J., Olshen, R., Stone, P., 1984. Classification and Regression Trees. Wadsworth, Belmont, CA.

Brown, M., Grundy, W., Lin, D., Cristianini, N., Sugnet, C., Furey, T., Ares, M., Haussler, J.D., 1997. Knowledge-based analysis of microarray gene expression data using support vector machines. Proceedings of the National Academy of Sciences USA 97 (1), 262–267.

Burges, C.J.C., 1998. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2 (2), 1–47.

Cristianini, N., Shawe-Taylor, J., 2000. Support Vector Machines. Cambridge University Press, Cambridge.

Devillers, J., 1999a. Genetic Algorithms in Molecular Modeling. Academic Press, New York.

Devillers, J., 1999b. Neural Networks and Drug Design. Academic Press, New York.

Duda, R., Hart, P., 1973. Pattern Classification and Scene Analysis. Wiley, New York.

Gini, G., Testaguzza, V., Benefenati, E., Todeschini, R., 1998. Hybrid toxicology expert system: architecture and implementation of a multi-domain hybrid expert system for toxicology. Chemometrics and Intelligent Laboratory System 43, 135–145.

Goldberg, D., 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading, MA.

Hawkins, D.M., Young, S.S., Rusinko, A., 1997. Analysis of a large structure–activity data set using recursive partitioning. Quantitative Structure–Activity Relationships 16, 296–302.

Hertz, J., Krogh, A., Palmer, R., 1991. Introduction to the Theory of Neural Computation. Addison-Wesley, Redwood City, CA.

Joachims, T., 1999. Making large-scale SVM learning practical. In: Schölkopf, B., Burges, C., Smola, A. (Eds.), Advances in Kernel Methods: Support Vector Learning. MIT Press, Cambridge, MA, pp. 169–184.

Joachims, T., 2000. Estimating the generalization peformance of a SVM efficiently. In: Langley, P. (Ed.), Proceedings of the Seventeenth International Conference on Machine Learning. Morgan Kaufmann, Los Altos, CA, pp. 431–438.

King, R.D., Muggleton, S., Lweis, R.A., Sternberg, M.J.E., 1992. Drug design by machine learning: the use of inductive logic programming to model the structure–activity relationships of trimethoprim analogues binding to dihydrofolate reductase. Proceedings of the National Academy of Sciences USA 89, 11322–11326.

Kövesdi, I., Dominguez, M., Ôrfi, L., Náray-Szabó, G., Varró, A., Papp, J., Mátyus, P., 1999. Application of neural networks in structure–activity relationships. Medicinal Research Reviews 19 (3), 249–269.

Lang, K., Waibel, A., Hinton, G., 1990. A time delay neural network architecture for isolated word recognition. Neural Networks 3, 33–43.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L., 1989. Backpropagation applied to handwritten zip code recognition. Neural Computation 1 (4), 541–551.

Manallack, D.T., Livingstone, D.J., 1999. Neural networks in drug discovery: have they lived up to their promise? European Journal of Medicinal Chemistry 34, 95–208.

Mercer, J., 1909. Functions of positive and negative type and their connection with the theory of integral equations. Philosophical Transactions of the Royal Society London A 209, 415–446.

Murthy, S.K., Kasif, S., Salzberg, S., 1994. A system for oblique induction of decision trees. Journal of Artificial Intelligence Research 2, 1–32.

Osuna, E., Freund, R., Girosi, F., 1997. Support vector machines: training and applications. AI Memo 1602, MIT.

Pomerleau, D., 1993. Knowledge-based training of artificial neural networks for autonomous robot driving. In: Connell, J., Mahadevan, S. (Eds.), Robot Learning. Kluwer Academic, Dordrecht, pp. 19–43.

Quinlan, J., 1992. C4.5: Programs for Machine Learning. Morgan Kaufmann, Los Altos, CA.

Rosenblatt, F., 1958. The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review 65, 386–408.

Sadowski, J., 2000. Optimization of chemical libraries by neural networks. Current Opinion in Chemical Biology 4, 280–282.

Schneider, G., 2000. Neural networks are useful tools for drug design. Neural Networks 13, 15–16.

SPSS, 1999. CLEMENTINE 5.1. URL: http://www.spss.com.

Vapnik, V., 1979. Estimation of Dependencies Based on Empirical Data. Nauka, Moscow.

Vapnik, V., 1998. Statistical Learning Theory. Wiley, New York.