Nonlinear Prediction of Quantitative Structure−Activity Relationships

Peter Tiňo*

School of Computer Science, Birmingham University, Birmingham B15 2TT, U.K.

Ian T. Nabney

Neural Computing Research Group, Aston University, Birmingham B4 7ET, U.K.

Bruce S. Williams and Jens Lösel

Pfizer Global Research and Development, Sandwich, Kent CT13 9NJ, U.K.

Yi Sun

Faculty of Engineering and Information Sciences, University of Hertfordshire, Hatfield, AL10 9AB, U.K.

Received November 7, 2003

Predicting the log of the partition coefficient *P* is a long-standing benchmark problem in Quantitative Structure−Activity Relationships (QSAR). In this paper we show that a relatively simple molecular representation (using 14 variables) can be combined with leading edge machine learning algorithms to predict logP on new compounds more accurately than existing benchmark algorithms which use complex molecular representations.

1. INTRODUCTION

The majority of pharmaceutical agents must cross a biological membrane to reach their site of action and to be available in a cellular environment. *Lipophilicity* of the "drug" molecule has a major impact upon its distribution and biological action.[1] Hence quantitative measures of lipophilicity are very important in the development of drug molecules.

The *partition coefficient* of a molecule is the ratio of its solubility in *n*-octanol to its solubility in water;[2] the logarithm of this quantity, LogP, is a well established measure of a compound's lipophilicity. In principle, the measurement of the equilibrium concentration of solute in the octanol and water phases, after shaking in a separatory funnel, is very simple, and since good measured values are always to be preferred over calculated ones, it would seem that there should be little need for a procedure to calculate them. However, in practice, measurement of LogP for large numbers of compounds is costly and time-consuming, and hence computational methods are employed to estimate or predict values where possible. In addition, it is valuable to have an estimate of lipophilicity before synthesizing novel compounds, and this can be only be done using a predictive model. Numerous methods for calculating LogP exist, mainly characterized as substructural and whole molecular approaches.[3]

The aims of this paper are to evaluate a novel encoding scheme for pharmaceutical molecules and assess its relevance for QSAR by applying a range of classical and more recently developed statistical and machine learning algorithms. The results are compared with state-of-the-art commercial programs. Throughout we take a *model-based* approach: our belief is that the encoding of the data has encapsulated sufficient prior understanding of the chemistry of the problem that a general-purpose statistical model will achieve the best accuracy, given the large amount of training data available.

In Section 2 we describe the algorithms that we applied to the QSAR problem. Performance measures are defined in Section 3 and the data sets in Section 4, where we also discuss the results. Finally, Section 5 draws together the main conclusions and points the way toward future work.

2. MODELING ALGORITHMS

The aim of QSAR is to find a function $q()$ which, given a structured representation of a molecule, predicts its activity:

$$activity = q(structure) \tag{1}$$

There are two main problems to solve:

1. the *representation* problem, i.e., how to encode molecules through the extraction and selection of structural features;

2. the *mapping* problem, i.e., determining the form of the function q and setting any free parameters so as to maximize the generalization performance of the model. For example, the weights attached to each input in a linear regression model are estimated from data.

In this paper, we will focus on the second of the two tasks, using a novel molecular encoding (an *interaction fingerprint*) developed at Pfizer. In this encoding, the molecule is coded using fourteen numerical descriptors based on a two-dimensional representation of the molecular structure. Values 1−10 are the InterAction Fingerprints, IAFs,[4] of the compounds. IAFs are the average counts for noncovalent interactions (strong, medium, weak hydrogen bonds, van der Waals and pi-interactions) around individual atom types as found in experimental structures deposited in the Cambridge Structure Database summed up over the whole molecule. Value 11 is the sum of volumes of Voronoi polyhedra which were used to determine the IAFs and is used as a measure

* Corresponding author e-mail: p.tino@cs.bham.ac.uk.

10.1021/ci034255i CCC: $27.50
Published on Web 07/28/2004

of size. Values 12−14 are halogen counts for fluorine, chlorine, and bromine.

The simplest predictive model always predicts a constant value, namely the mean $E(\boldsymbol{t}^{trn})$ of the training set targets, no matter what test input it receives. We will refer to this model as the *Naive* predictor, and the this model provides a lower bound on useful performance levels.

Besides the *Naive* model, we considered seven model classes that can be divided into two categories: global and local models.

Global models use a single model for the problem which covers the entire input space.

Local models use a combination of models, each of which covers a smaller part of the input space.

There are two reasons why a local model might be expected to perform better than a global model:

1. The form of the mapping that we are trying to learn may be different for different regions of the input space. For example, for some compound classes the relationship between inputs and targets may be linear, while for other compound classes it may be quadratic.

2. The component models can be simpler (to match the complexity of the local mapping) which may make them easier to fit. For example, the parameters in a linear regression model can be determined using linear algebra, while the parameters in a neural network are determined by many iterations of a nonlinear optimization algorithm.

The first of these reasons is particularly relevant to chemical structures.

If the input space has an obvious decomposition, then it may be possible to determine the structure of the local model by hand. However, this is rarely the case, and so we used models which discovered the decomposition automatically as part of the learning process.

The fundamental task in estimating the parameters of a model is to try to fit the underlying generator of the data. This can be measured by how well the model predicts target values on *new* data. Given a particular model class $\mathcal{M}$ there usually will be many candidate models $M \in \mathcal{M}$ that can be potentially used as predictors for our task. In general, complex candidates with a large number of free parameters will be able to model the training data very well but will perform poorly on an independent test set. This happens because, loosely speaking, complex models are overly powerful and "read too much into the training data". They tend to concentrate on misleading details of noise in the data and introduce rather arbitrary complex response patterns for inputs not covered by the training set. This phenomenon is often referred to as *overfitting* the (training) data.

There are several approaches to deal with the phenomenon of overfitting. One of the most popular within the machine learning community consists of creating a *validation set* that is disjoint from both the training and test sets. When reasoning about what candidate M from the model class $\mathcal{M}$ to use as a representative of $\mathcal{M}$, several candidates M_1, M_2, ..., $M_C \in \mathcal{M}$ of increasing model complexity are trained on the training set and tested on the validation set. The candidate with the minimal validation set error is then selected as a representative of $\mathcal{M}$ and its generalization error is estimated on the test set. Such a validation set approach to model selection is appropriate in situations when there is enough data at our disposal, which is the case here.

It is helpful to define the statistical pattern recognition framework that we will use in this paper (see also ref 5). The most general information about a target variable t for inputs $\boldsymbol{x}$ is given by the conditional density $p(t|\boldsymbol{x})$. The variable is modeled as the output of a deterministic function y with parameters $\boldsymbol{w}$ plus random noise ϵ:

$$t = y(\boldsymbol{x};\boldsymbol{w}) + \epsilon \tag{2}$$

Assuming that each data point $(\boldsymbol{x}_n, t_n)$ is drawn independently from a fixed distribution and that the noise model ϵ has a Gaussian distribution with zero mean and constant variance σ^2, the conditional density has the following form:

$$p(t|\boldsymbol{x}) = \frac{1}{(2\pi\sigma^2)^{1/2}}\exp\left(-\frac{\{y(\boldsymbol{x};\boldsymbol{w}) - t\}^2}{2\sigma^2}\right) \tag{3}$$

If we take logs and ignore terms that do not depend on the network function y, we arrive at the sum-of-squares error function

$$E_D = \frac{1}{2}\sum_{n=1}^{N}\{y(\boldsymbol{x}_n;\boldsymbol{w}) - t_n\}^2 \tag{4}$$

In the experiments reported here, all models had fourteen inputs and a single output.

2.1. Global Models. Obviously, the *Naive* predictor is a trivial global model. We considered four more global model classes.

2.1.1. Linear Regression − LR. The output y is a linear combination of input values $\boldsymbol{x}$

$$y = y(\boldsymbol{x};\boldsymbol{w}) = \sum_{i=1}^{d} w_i x_i + b \tag{5}$$

where d is dimensionality of the input space and $\boldsymbol{w} = (w_1,..., w_d, b)$ is the parameter vector.

We set the weights so that the sum-squared error function is minimized on a training data set. In the case of linear regression, because E is a quadratic function of the weights, this can be done using the pseudoinverse of the data matrix, a standard technique from linear algebra.[6]

Linear regression can be generalized by including nonlinear functions of predictors, while keeping the models linear in parameters. We augment the linear regression model either with only squares x_i^2 of predictors x_i

$$y = \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} w_i^{(sq)} x_i^{\,2} + b \tag{6}$$

or with all quadratic terms $x_i \cdot x_j$ constructed from the predictors

$$y = \sum_{i=1}^{d} w_i x_i + \sum_{j \le i}^{d} w_{ij}^{(q)} x_i x_j + b \tag{7}$$

Free parameters of such models, b, w_i, $w_i^{(sq)}$, $w_{ij}^{(q)}$, can still be estimated using tools from linear algebra. We refer to the former and latter models as **LSR** and **LQR**, respectively.

2.1.2. Multilayer Perceptron − MLP. The multilayer perceptron is a *nonlinear* regression model. It is probably the most widely used architecture for practical applications

of neural networks. In most cases the network consists of two layers of adaptive weights with full connectivity between inputs and hidden units and between hidden units and outputs. It is capable of *universal approximation* in the sense that it can approximate to arbitrary accuracy any continuous function from a compact region of input space provided the number of hidden units is sufficiently large and provided the weights and biases are chosen appropriately.[7−9] In practice, this means that provided there is enough data to estimate the network parameters, an MLP can model any smooth function.

The first layer of the network forms N_{hid} linear combinations of these inputs to give a set of intermediate activation variables which are then transformed by the nonlinear activation functions of the hidden layer, here chosen to be the tanh function, to give the hidden unit outputs z_j

$$z_j = \tanh\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + b_j^{(1)}\right) \; j = 1, \ldots, N_{\text{hid}} \tag{8}$$

Here $w_{ji}^{(1)}$ represents the elements of the first-layer weight matrix and $b_j^{(1)}$ are the bias parameters associated with the hidden units.

The z_j are then transformed by the second layer of weights and biases to give the network output y

$$y = \sum_{j=1}^{N_{\text{hid}}} w_j^{(2)} z_j + b^{(2)} \tag{9}$$

We make the same assumption about the noise model as for linear regression and therefore use the sum-of-squares error function given by (4). Because of the nonlinear relationship between the network weights and the output, it is no longer possible to use a matrix pseudoinverse to find the weights directly. However, the back-propagation algorithm can be used to compute the error gradients $\partial E/\partial w_i$ efficiently,[5] and these gradients can be used to adjust the weights to minimize E using a nonlinear optimization routine. In the experiments reported in this paper, all the networks were trained using the scaled gradient method.[10]

We considered N_{hid} from the range $N_{\text{hid}} \in \{1, 2, 3, 5, 10, 15, 20, 30\}$. To stabilize the performance, for each number of hidden units N_{hid}, we trained a *committee* of $N_{exp} = 10$ networks of the same topology. Given a test input, the predictive output y was obtained as the mean of the outputs y^j of the committee members:

$$y = \frac{1}{N_{exp}} \sum_{j=1}^{N_{exp}} y^j \tag{10}$$

The number of hidden neurons was determined by the performance on the validation set: we picked N_{hid} with the smallest validation set MSE. It can be shown (see ref 5, Chapter 9) that the expected error of the committee is not greater than the average error of the networks that make up the committee. In practice, the committee error is often better than that of any individual network.

2.1.3. MLP with Automatic Relevance Determination − MLP-ARD. Selecting relevant input variables from a large set of possibilities is a common problem in applications of pattern analysis. Selecting variables based on the magnitude of their correlation with the target variable is based only on linear relationships; frequently the importance of inputs is only revealed in a nonlinear mapping. What is required is a way to determine the importance of each input to a trained model.

We have used a Bayesian approach to this problem,[11,12] which also has the benefit of *regularizing* the model. In the Bayesian approach, we start with a *prior* probability distribution $p(\boldsymbol{w})$ which expresses our knowledge of the parameters before the data is observed. Typically this is quite a broad distribution, reflecting the fact that we only have a vague belief in a range of possible parameter values (for example, that the magnitude of the parameters should usually be less than 10). Once we have observed the data, Bayes' theorem

$$p(\boldsymbol{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})}{p(\mathcal{D})} \tag{11}$$

(where $p(\boldsymbol{w})$ is the prior, $p(\mathcal{D}|\boldsymbol{w})$ is the data set likelihood, and $p(\mathcal{D})$, known as the *evidence*, is a normalization factor that ensures that the posterior integrates to 1) can be used to update our beliefs, and we obtain the *posterior* probability density $p(\boldsymbol{w}|\mathcal{D})$. Since some parameter values are more consistent with the data than others, the posterior distribution is concentrated on a smaller range of values than the prior distribution.

The prior probability distribution for the weights of a neural network should embody our prior knowledge on the sort of mappings that are "reasonable". In general, we expect the underlying generator of our data sets to be smooth, and the network mapping should reflect this belief. A neural network with large weights will usually give rise to a mapping with large curvature, and so we favor small values for the network weights. The requirement for small weights suggests a Gaussian prior distribution with zero mean of the form

$$p(\boldsymbol{w}) = \frac{1}{Z_W(\alpha)} \exp\left(-\frac{\alpha}{2}\left\|\boldsymbol{w}\right\|^2\right) \tag{12}$$

where α represents the *inverse* variance of the distribution and the normalization constant is given by

$$Z_W(\alpha) = \left(\frac{2\pi}{\alpha}\right)^{W/2} \tag{13}$$

where W is the number of weights stored in the weight vector $\boldsymbol{w}$.

Because α is a parameter for the distribution of other parameters (weights and biases), it is known as a *hyperparameter*. Ignoring the normalization constant, which does not depend on the weights, this prior is equivalent to a weight error term (after taking the negative log) of the form

$$\alpha E_W = \frac{\alpha}{2}||\boldsymbol{w}||^2 = \frac{\alpha}{2}\sum_{i=1}^{W} w_i^2 \tag{14}$$

This error term *regularizes* the weights by penalizing overly large magnitudes. The error function E used in training is found by taking the negative log of (11); ignoring constants, it has the form

$$E = \alpha E_W + \beta E_D \tag{15}$$

where E_D is as in (4), and $\beta = 1/\sigma^2$ is the inverse variance of the likelihood (3).

It is helpful to generalize this notion to multiple hyperparameters α_1, ..., α_g corresponding to groups of weights $\mathcal{W}_1$, ..., $\mathcal{W}_g$. For Automatic Relevance Determination (ARD), we associate a separate hyperparameter with each input variable; this represents the inverse variance of the prior distribution of the weights fanning out from that input. During Bayesian learning it is possible to modify the hyperparameters; for example, using the evidence procedure, we find their optimal value, subject to some simplifying approximations to make the analysis tractable. Because hyperparameters represent the inverse variance of the weights, a small hyperparameter value means that large weights are allowed, and we can conclude that the corresponding input is important. A large hyperparameter value means that the weights are constrained near zero, and hence the corresponding input is less important. Additional hyperparameters may be used for other groups of weights in the network.

2.1.4. Gaussian Process − GP. Gaussian processes (GPs) are a relatively recent development in nonlinear modeling,[13] though they have a longer history in spatial statistics, where the technique is also known as "kriging". Gaussian processes are particularly suited to regression problems since in these circumstances we can perform the first level of Bayesian inference (computing the posterior distribution of the parameters) analytically.

A *Gaussian process* is a stochastic process $Y(\boldsymbol{x})$ where every joint density function is Gaussian and is therefore defined completely by its mean and covariance. For simplicity, we will consider only Gaussian processes with zero mean. The covariance of $Y(\boldsymbol{x})$ and $Y(\boldsymbol{x}')$ is usually defined by a function $C(\boldsymbol{x},\boldsymbol{x}')$.

Recall that the training data set consists of ordered pairs $(\boldsymbol{x}_1^{trn}, t_1^{trn})$, ..., $(\boldsymbol{x}_{N_{trn}}^{trn}, t_{N_{trn}}^{trn})$. Now suppose that t_i^{trn} is a sample from a random variable $T(\boldsymbol{x}_i^{trn})$. To make a prediction T^* at a new input $\boldsymbol{x}^*$ we need to compute the conditional distribution $p(T^*| T_1, ..., T_{N_{trn}})$. Since our model is a Gaussian process, this distribution is also Gaussian and is completely specified by its mean and variance. Let $\boldsymbol{K}$ denote the covariance matrix of the training data, $\boldsymbol{k}$ denote the $N_{trn} \times 1$ covariance between the training data and $\boldsymbol{T}^*$, and k^* denote the variance of T^*. Then $\boldsymbol{K}_+$, the $(N_{trn} + 1) \times (N_{trn} + 1)$ covariance matrix of $(T_1, T_2, ..., T_{N_{trn}}, T^*)$, can be partitioned

$$\boldsymbol{K}_+ = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{k} \\ \boldsymbol{k}^T & k^* \end{bmatrix} \tag{16}$$

The conditional mean and variance at $\boldsymbol{x}^*$ are given by

$$E[T^*] = \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{t}^{trn} \tag{17}$$

$$\mathrm{var}[T^*] = k^* - \boldsymbol{k}^T \boldsymbol{K}^{-1} \boldsymbol{k} \tag{18}$$

We use the mean as our prediction, while the covariance can be used to compute error bars.

The covariance function is defined by the spherical Gaussian kernel of width σ^2

$$C(\boldsymbol{x},\boldsymbol{x}') = \exp(||\boldsymbol{x} - \boldsymbol{x}'||/2\sigma^2) \tag{19}$$

In general, larger widths imply "smoother" regression functions. For Gaussian kernels and normalized input data, for which the standard deviation is equal to 1 in all dimensions, the widely accepted rule of thumb is to make the kernel width approximately equal to the input dimensionality d (in our case $d = 14$). We considered widths from the set

$$\sigma^2 = \{6, 8, 10, 12, 14, 16, 18, 20, 25, 30, 40, 50, 60\}$$

The "optimal" kernel width was determined by the validation set performance. We report the GP test set performance using σ^2 with the smallest validation set MSE.

It is well-known that GP prediction scales badly with the number of training examples. This is because $\boldsymbol{K}$ is an $N_{trn} \times N_{trn}$ matrix, and the matrix inversion needed in (17) is $O(N_{trn}^3)$. This is problematic in this application, as we need to use large training sets to obtain good generalization performance. We therefore used with a sparse on-line version of GP prediction developed in ref 14.

2.2. Local Models. We considered three kinds of local model, namely K-nearest-neighbor regression, hierarchical mixture of experts, and regression trees.

2.2.1. *K*-Nearest-Neighbor Regression − KNNR. *K*-nearest-neighbor regression (KNNR) can be considered a committee of *K* experts of a special type: given a test input $\boldsymbol{x}$ the prediction y^j of the jth expert is equal to the target value $t_{j(x)}^{trn}$ of the training pair $(\boldsymbol{x}_{j(x)}^{trn}, t_{j(x)}^{trn})$ whose input $\boldsymbol{x}_{j(x)}^{trn}$ is the jth closest point to $\boldsymbol{x}$ among all the training inputs $\{\boldsymbol{x}_1^{trn}, ..., \boldsymbol{x}_{N_{trn}}^{trn}\}$. The output of the model is therefore the average target value for the *K* training data points closest to $\boldsymbol{x}$.

In the experiments, we varied the number of neighbors, *K*, between 1 and 15, and the "optimal" *K* was determined on the validation set by selecting *K* from the model with the minimal validation set MSE.

2.2.2. Hierarchical Mixtures of Experts − HME. The HME model[15] divides the input space into a nested set of regions. In each region a simple surface (a linear hyperplane) is fitted to the data. The regions have "soft" boundaries, which means that points belong to more than one region. Figure 1 shows the architecture of the HME model. The architecture consists of a binary tree (the generalization to more branches at each node is trivial) consisting of gating networks and experts. These networks receive the input vector $\boldsymbol{x}$ and produce scalar outputs p_i with the property that $p_i \geq 0$ and $\sum_i p_i = 1$. The output is a sum of the expert predictions weighted by the p_i. To form a deeper tree, each expert is expanded recursively into a gating network and a set of subexperts.

The role of the gating networks is to partition the input space so that each expert only has to model a small region. In our experiments, all the local models were linear regression predictors with a Gaussian noise model. This model gives rise to a well-defined conditional probability $p(t|\boldsymbol{x})$, and all the parameters can be adjusted using the negative log likelihood of the training data.

The hierarchy can be specified by the depth *D* of the hierarchical tree and the branching factor *BF*—the number of children of each internal node. The number of local linear regression experts in the hierarchy, i.e., the number of leaves of the hierarchical tree, is given by $(BF)^{D-1}$.

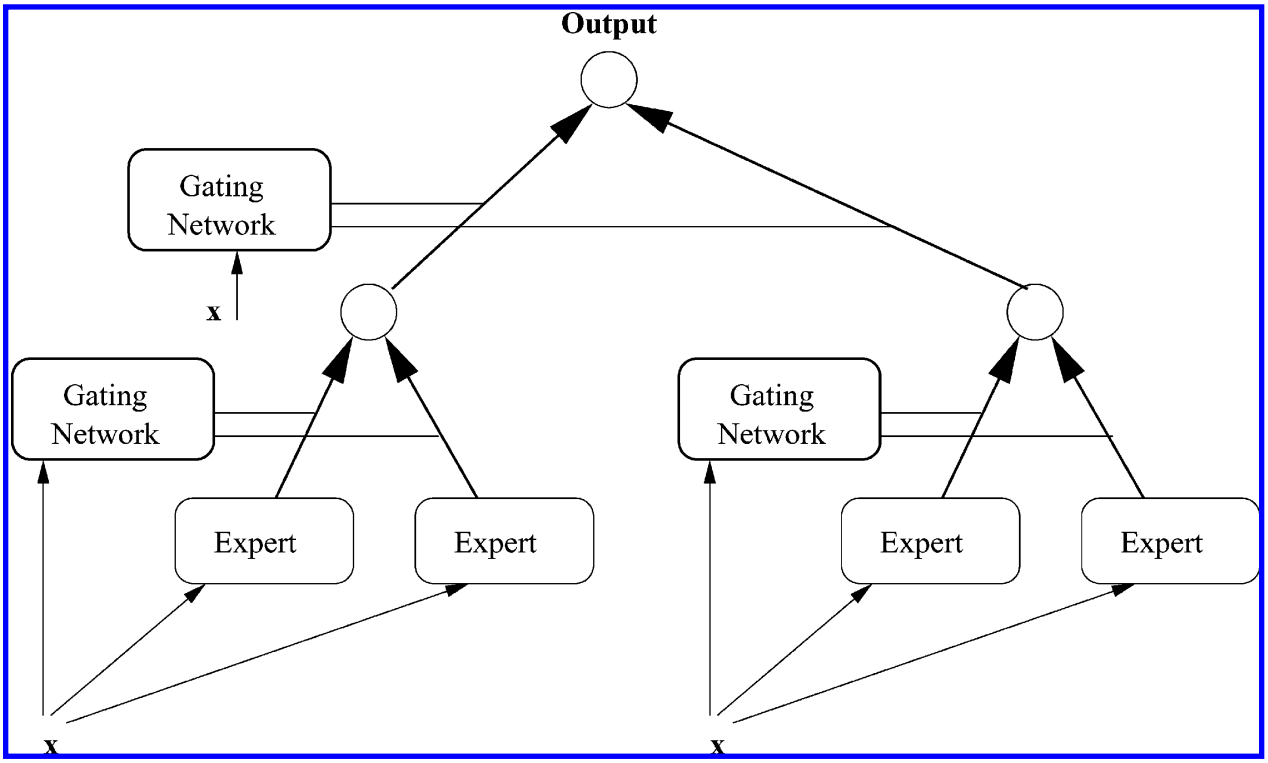

Figure 1. Architecture of a two-level HME model. The gating networks act as swtiches between the experts.

We considered the following hierarchies (*BF*, *D*): simple mixtures (2, 1), (4, 1); depth-2 hierarchies (2, 2), (3, 2); and deeper hierarchies (2, 3), (3, 3), (2, 4), (3, 4). The hierarchy used on the test set was determined by the minimal validation set MSE.

2.2.3. Regression Trees − RT. Regression trees (RTs) are computationally efficient nonparametric models that constitute a good compromise between comprehensibility and predictive accuracy. A regression tree consists of a hierarchy of nodes. With the exception of the bottom nodes (leaves) of the tree, each node contains a logical test on one of the input variables x_i, $i \in \{1, ..., d\}$. Each test has the form [Variable Operator Value] (e.g. $x_2 < 5.7$) and has two possible outcomes, *true* or *false*. Any path from the top node (root) to a leaf can be seen as a conjunction (i.e. logical *and*) of logical tests on the input coordinates. These conjunctions are logical representations of a partition of the input space. Each leaf contains a local predictive model, which in the case of standard RTs is simply a constant value. The local model associated with a leaf operates over a corresponding region of the input space that is defined by the conjunction.

However, constant values in the leaves lead to a regression surface that is not continuous (in fact, a step function). A smoother model is achieved by allowing nonconstant leaf models, e.g. linear functions of inputs $\boldsymbol{x}$.[16,17] An example of a regression tree with linear models in the leaves is shown in Figure 2.

We trained a large collection of RTs with constant and linear regression models in the leaves using the system RT4.1.[17] The individual RT constructions are controlled by a set of hyperparameters determining the type of local regression models and other model specific entities, such as the pruning methods, etc.

The construction of RTs involved a pruning mechanism, where for each hyperparameter value a sequence of trees is generated (using a method called *lowest statistical support*[17]), and for every regression tree in that sequence its generalization error is estimated via a 5-fold cross validation on the training set. The representative RT for each particular hyperparameter setting is the one with the lowest estimated generalization error.

The hyperparameter setting to be used on the test set was determined by the minimal validation set MSE.

We characterize the RTs by the local regression model (constant(C)/linear regression(LR)), the overall number of nodes *N*, and the number of leaves *L*. For example, the tree in Figure 2 is characterized by *(LR, N=5, L=3)*.

3. EXPERIMENTS

3.1. Performance Measures. We evaluate the test set model performance via two related measures, namely mean square error and percent improvement over the *Naive* model.

Suppose we are given N_{trn} and N_{tst} training and test input-target pairs $(\boldsymbol{x}_n^{trn}, t_n^{trn})$ and $(\boldsymbol{x}_n^{tst}, t_n^{tst})$, respectively. Model prediction, given a test input $\boldsymbol{x}_n^{tst}$, is denoted by $\hat{t}_n$.

Mean squared error measures the average squared difference between model predictions $\hat{t}_n$ and the corresponding targets t_n^{tst}

$$\mathrm{MSE} = \frac{1}{N_{tst}} \sum_{n=1}^{N_{tst}} (\hat{t}_n - t_n^{tst})^2 \tag{20}$$

It is useful to report improvement of MSE relative to the baseline $\mathrm{MSE}_{\mathrm{naive}}$ of the *Naive* predictor always predicting mean

$$\hat{t}_{\mathrm{naive}} = \frac{1}{N_{trn}} \sum_{n=1}^{N_{trn}} t_n^{trn} \tag{21}$$

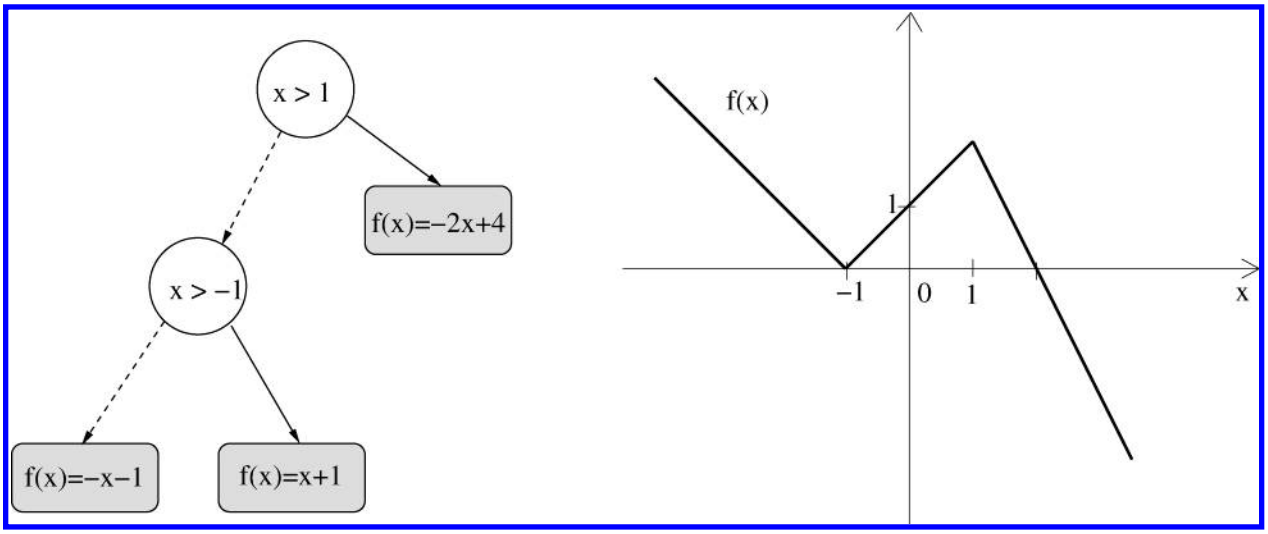

Figure 2. Example of regression tree with linear models in the leaves. The tree defines a piece-wise linear function on real line. Arcs corresponding to *true* and *false* decisions are shown as solid and dashed lines, respectively.

of the *training* targets

$$\mathrm{MSE}_{\mathrm{naive}} = \frac{1}{N_{tst}} \sum_{n=1}^{N_{tst}} (\hat{t}_{\mathrm{naive}} - t_n^{tst})^2 \qquad (22)$$

The degree of improvement (expressed in percentages) of the model over the *Naive* predictor is quantified by the *improvement over Naive* (ION) measure

$$\mathrm{ION} = \frac{\mathrm{MSE}_{\mathrm{naive}} - \mathrm{MSE}}{\mathrm{MSE}_{\mathrm{naive}}} \cdot 100\% \qquad (23)$$

ION is closely related to squared multiple correlation that uses variance in test targets instead of $\mathrm{MSE}_{\mathrm{naive}}$.[Variance in test targets can be viewed as MSE of a simple predictor always predicting the mean of the test targets. Usually the variance over the training targets is close to that of the test targets, in which case squared multiple correlation can be interpreted as measuring ION.]

3.2. Data Sets. We worked with two data sets:

• *Data set I* is a set of 6912 compounds together with their LogP values that is freely available on the Internet. 10% of the set was used as a test set, and, when needed, another 10% was set apart as a validation set for model selection. The remaining data was used as a training set.

• *Data set I+II* consists of Data set I augumented by *Data set II*, a new set of 226 compounds whose LogP values were measured at Pfizer. Data set II served as a completely blind test set for models trained on the whole of Data set I. [When needed, 10% of Data set I was set apart as a validation set for model selection.]

Compounds in both data sets were coded by Pfizer as 14-dimensional real-valued vectors. For the experiments involving neural networks, the toolbox[18] was used.

The experimental results for sets *Data set I* and *Data set I+II* are presented in Tables 1 and 2, respectively.

3.3. Discussion. For laboratory chemists the key question is which method of calculating logP gives the most accurate results. It is vital that this accuracy is evaluated on unseen data, since there is no point in using a predictive model if the true value has already been measured. We compared our results for *Data set I* with those of four commonly used commercial methods for LogP prediction: ClogP,[19] (PC Models Version 4.71 implemented in the Daylight System), AlogP,[20,21] (implemented in Cerius2) a multilinear regression of IAFs, and MlogP[22] (an in-house implementation). These systems are based on much more complex molecular representations than the 14 variables used in our models. For example, the ClogP algorithm breaks down a molecule into fragments, each of which is assigned a value in a weighted sum. Correction factors (e.g. bond types, branching at isolated carbon atoms) and interaction factors (e.g. aliphatic proximity and π-bonds) are then used to modify the initial estimate.

Table 1. Experimental Results on Data Set I

	data set I		
model class	MSE	ION	selected model
Naive	2.690	0	Naive
LR	0.799	70.3	LR
LSR	0.717	73.3	LSR
LQR	0.740	72.5	LQR
MLP	0.641	76.2	$N_{hid} = 15$
MLP-ARD	0.611	77.3	$N_{hid} = 15$
GP	0.601	77.7	$\sigma^2 = 40$
KNNR	0.918	65.9	K=4
RT	0.737	72.6	LR, N=21, L=11
HME	0.658	75.5	BF=2, D=3

Table 2. Experimental Results on Data Set I+II

	data set I+II		
model class	MSE	ION	selected model
Naive	4.075	0	Naive
LR	1.190	70.8	LR
LSR	1.084	73.4	LSR
LQR	1.163	71.5	LQR
MLP	1.075	73.6	$N_{hid} = 15$
MLP-ARD	1.119	72.5	$N_{hid} = 15$
GP	1.074	73.6	$\sigma^2 = 50$
KNNR	1.738	57.3	K=6
RT	0.979	76.0	LR, N=21, L=11
HME	0.953	76.6	BF=2, D=3

The results for the benchmark algorithms on Data sets I and I+II are given in Table 3. The results seem to show a clear advantage for the ClogP method on Data set I, but it has to be stressed that these compounds have been used as

Table 3. Results of Four Commonly Used Methods for LogP Prediction

	data set I		data set I+II	
method	MSE	ION	MSE	ION
ClogP	0.175	93.5	1.023	74.9
AlogP	0.550	79.5	1.164	71.4
MlogP	2.109	21.6	1.088	73.3
IAF	1.318	51.0	1.077	73.6

part of the training set for both the ClogP method and the multilinear regression of IAFs. Hence they should be considered as training set performance, rather than true generalization performance. The test results on Data set I+II are based entirely on 226 compounds of pharmaceutical interest that were not used for training any of the methods in this paper (or the benchmark algorithms) and are thus much more representative of the performance of each technique on new data. A comparison of the ION values with Table 2 shows that hierarchical mixtures of experts and regression trees outperformed all the standard methods, while the multilayer perceptron and Gaussian processes performed better (all equal) than all but ClogP. The large worsening in ION on the test set for ClogP is a sign that the model is significantly overfitted to its training data. This can be compared with the machine learning algorithms used in this paper, whose performance on the test set is comparable to that on the training set.

4. CONCLUSIONS

1. The novel compound representations we used are relatively simple yet can lead to results comparable to those of commercial state-of-art products operating on more complicated molecular representations. This is particularly clear when comparing the generalization performance on previously unseen compounds; this is precisely the situation of most value to working chemists.

2. Advanced regression tools such as hierarchical mixtures of experts (HME) operating on the novel representation achieve results comparable with those of more complex and expensive state-of-art products.

3. Local regression models tend to out-perform global regression models.

There is clearly scope to improve these results further, both by increasing the size of feature set used as inputs and by developing further the regression models. We have a particular interest in taking the local modeling further by combining it with hierarchical visualization techniques.[23] Preliminary results with this combined model are very promising.

REFERENCES AND NOTES

(1) Lipinski, C.; Lombardo, F.; Dominy, B.; Feeney, P. *Adv. Drug Delivery Rev.* **1997**, *23*, 3−25.
(2) Sangster, J. *Octanol−Water Partition Coefficients: Fundamentals and Physical Chemistry;* Wiley Series in Solution Chemistry, Wiley, 1997; Vol. 2.
(3) Mannhold, R.; van de Waterbeemd, H. *J. Comput.-Aided Mol. Des.* **2001**, *15*, 337−354.
(4) Lösel, J. Interaction Fingerprints − a new descriptor for noncovalent interactions around functional groups. In *MGMS*; 1998.
(5) Bishop, C. M. *Neural Networks for Pattern Recognition;* Oxford University Press: 1995.
(6) Golub, G. H.; van Loan, C. F. *Matrix Computations;* Johns Hopkins University Press: Baltimore, 1996.
(7) Hornik, K.; Stinchcombe, M.; White, H. *Neural Networks* **1989**, *2*, 359−366.
(8) Stinchecombe, M.; White, H. Universal approximation using feed-forward networks with nonsigmoid hidden layer activation functions. In *Proceedings of the International Joint Conference on Neural Networks*; IEEE: San Diego, 1989; Vol. 1.
(9) Hornik, K. *Neural Networks* **1991**, *4*, 251−257.
(10) Møller, M. *Neural Networks* **1993**, *6*, 525−533.
(11) MacKay, D. J. C. *Neural Comput.* **1992**, *4*, 448−472.
(12) Neal, R. M. *Bayesian Learning for Neural Networks;* Lecture Notes in Statistics 118; Springer-Verlag: New York, 1996.
(13) Williams, C. K. I. Prediction with Gaussian Processes: from Linear Regression to Linear Prediction and Beyond. In *Learning and Inference in Graphical Models*; Jordan, M. I., Ed.; Kluwer: Dordrecht, 1998.
(14) Csató, L.; Opper, M. *Neural Comput.* **2002**, *14*, 641−668.
(15) Jacobs, R. A.; Jordan, M. I.; Nowlan, S. J.; Hinton, G. E. *Neural Comput.* **1991**, *3*, 79−87.
(16) Torgo, L. Functional models for regression tree leaves. In *Proceedings of the International Conference on Machine Learning (ICML-97)*; Morgan Kaufmann Publishers: 1997.
(17) Torgo, L. *Inductive Learning of Tree-vased Regression Models*; Thesis, University of Porto, 1999.
(18) Nabney, I. T. *Netlab: Algorithms for Pattern Recognition;* Advances in Pattern Recognition Springer-Verlag: London, 2002.
(19) Hansch, A.; Leo, A. *Substituent Constants for Correlation Analysis in Chemistry and Biology;* Wiley-Interscience Wiley: New York, 1979.
(20) Ghose, A.; Crippen, G. *J. Comput. Chem.* **1986**, *7*, 565−577.
(21) Viswanadhan, V.; Ghose, A.; Revankar, G.; Robins, R. *J. Chem. Inf. Comput. Sci.* **1989**, *29*, 163−172.
(22) Moriguchi, I.; Hirono, S.; Liu, Q.; Nakagome, I.; Matsushita, Y. *Chem. Pharm. Bull.* **1992**, *40*, 127−130.
(23) Tiňo, P.; Nabney, I. T. *IEEE J. Pattern Anal. Machine Intelligence* **2002**, *24*, 639−656.

CI034255I