University Freiburg

Faculty for Applied Sciences

A Quantitative Extension to the Lazar Algorithm for the Prediction of Chemical Properties

Thesis to obtain the degree of Diplominformatiker by

Andreas Maunz

Student ID 1324232

Sundgauallee 26-00-01, D-79110 Freiburg

E-Mail: maunza@fdm.uni-freiburg.de

Supervised by the Chair for Bioinformatics

Prof. Dr. Rolf Backofen

June 22, 2007

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum Unterschrift

Abstract

The reliable computer-based prediction of toxicological activities of chemicals is a valuable achievement. One can spare costly and ethically problematic animal testing, and it is much faster than in-vitro methods. The problem can be approached with expert systems, full featured (Q)SAR models or other strategies.

This work presents a robust instance-based approach using linear fragments of chemical compounds as descriptors. Predictions are derived using distance-weighted k-nearest-neighbour techniques and are assigned confidence values. The algorithm employs data mining techniques such as significance tests.

Three models have been developed and evaluated using leave-one-out crossvalidation for two popular, publicly available databases, namely a simple median prediction, and a multilinear model with and without prior principal components analysis.

The multilinear approach yielded R^2 values of 70% for more than half of the data set and up to 83% for predictions with a higher confidence level. These results put Lazar on a competitive level.

Contents

1 Introduction

This section introduces the prediction of chemical properties as an application of machine learning methods, specifically instance-based learning methods.

1.1 Predictive Toxicology

1.1.1 Overview

The task of *Predictive Toxicology* is to predict toxic effects from chemical and biological information, usually with the help of a computer program. It is therefore influenced by biology, chemistry and computer science (figure 1).

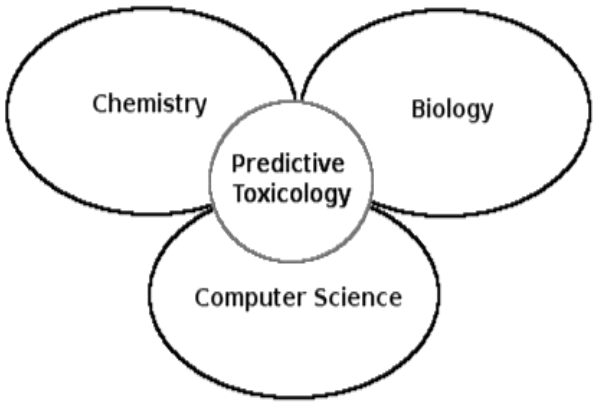


Figure 1: Predictive Toxicology and neighbouring disciplines

Recently, *Artificial Intelligence* methods have been used in predictive toxicology. The data driven approach presented in this work employs *Machine Learning* techniques. Machine learning, a broad subfield of artificial intelligence, is concerned with computer programs that improve their behaviour through learning over time and/or from training data. Machine learning uses techniques such as statistical significance tests, clustering and classification.

In a more general sense, machine learning is a *Data Mining* technique

to achieve the goal of knowledge discovery in massive amounts of data - amounts that cannot be handled by humans which is also often due to its high dimensionality.

More formally, machine learning methods are used to learn a model from training data that is able to predict the activity or impact of a chemical structure towards a specific toxicological endpoint.

Experimental data from in-vitro and in-vivo bioassays can be used as training data in predictive toxicology. Toxic effects are represented as qualitative (e.g. carcinogenicity classifications), or as quantitative (e.g. LC_{50}) values, or clinical records of patients that allow for classification. Compounds are described by structural information or physicochemical properties (e.g. the octanol-water partition coefficient or quantum-chemical parameters). The idea of predictive toxicology is to find regularities between those properties and toxic activities and use them for the prediction of untested compounds.

In learning, the model is modified according to some optimisation criterion, for example maximum likelihood statistics, which fit best a given set of training data, thereby minimising the error of the model.

> "The primary aim of predictive toxicology is, of course, the prediction of toxic activities of untested compounds. This enables chemical and pharmaceutical companies, for example, to evaluate potential side effects of candidate structures even without synthesizing them. The same feature is also attractive for governmental authorities that have to deal with compounds with incomplete toxicity information. [...]
> Many predictive toxicology systems are capable to provide a rationale for their predictions. If the prediction is based on chemical substructures and/or chemical properties, it is straightforward to use this information for the design of less dangerous, but equally efficient compounds." [Hel04]

1.1.2 Molecular Descriptors

The toxicological phenomenon to be analyzed determines to a great extent what information should be incorporated into the training data. Often, in

biological settings, receptor-ligand processes play a key role. If the ligand is present, the receptor initiates a cellular response. Chemically speaking, this mechanism is mediated by interactions between the molecules that the participants consist of. Therefore, the chemical structure is naturally informative when it comes to analyse such processes.

There is a wide variety of structural descriptors that could be used, for example 1D, 2D and 3D molecular descriptors as well as fragment-based ones. There is no "universal set" of descriptors which can be generally applied. The search for appropriate descriptors is guided by the need for features that are expressive enough, but at the same time as simple as possible. In other words, they should be relevant and interpretable for the toxicological mechanism to be described, but not too computationally expensive.

Expressive also means that making small changes to the descriptors of a molecule only causes a small change in the activity of the molecule. This relation is crucial for the use of descriptors, because it allows the interpolation of activities of similar compounds to derive a prediction. Algorithms that learn a model from training data (including Lazar) rely on that principle.

1.2 Instance-Based Learning

The following introduces k-nearest-neighbour techniques, which is an *Instance-Based Learning* technique (also called *Lazy Learning* in contrast to *Eager Learning* methods [Mit97]).

Often, when learning a *Target Function* by interpolating from training data in order to predict unknown instances, a global model is learned that best fits the entire dataset. This approach is called eager learning, because every piece of information is incorporated into the model before making any prediction. With eager learning the model is a fixed function, and for every unknown instance the prediction is computed by simply evaluating the function on this instance. This is different from Lazy Learning methods, where training data is simply stored. No global model is learned beforehand, rather, for every query a new individual prediction is derived. We now investigate two common lazy learning methods that Lazar uses.

Assume all instances in a training database correspond to points in the

n-dimensional feature space $\mathbb{R}^n$. Every instance x can then be described by the feature vector $\langle a_1(x), \ldots, a_n(x)\rangle$ and we know the function values $f(x)$ for all x of the otherwise unknown target function f.
The notion of distance between points is crucial for the instance-based learning methods presented here. For example, the naturally induced euclidian distance between two points x_i and x_j is defined as $d(x_i, x_j) = \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2}$, but other distance measures are possible, too.

1.2.1 k-Nearest-Neighbour Prediction

The k *nearest neighbours* (or *neighbours* for short) to the query instance x_q are the k database instances with the smallest distance to x_q. The k-nearest-neighbour prediction (or *knn prediction* for short) is defined as

$$\widehat{f}(x) = \frac{\sum_{i=1}^{k} f(x_i)}{k}, \tag{1}$$

i.e., the mean of the function values of the neighbours. One could also use the median which is similar to the mean but more robust against extreme values (see section 2). In case that the neighbours are distributed very unequally around the query structure a distance-weighting is favorable. This can be achieved by a slight generalization of equation (1):

$$\widehat{f}(x) = \frac{\sum_{i=1}^{k} w_i * f(x_i)}{\sum_{i=1}^{k} w_i}, \tag{2}$$

where w_i is a function of $d(x_i, x_q)$). For example, in the euclidian distance case one could use $w_i = \frac{1}{d(x_q, x_i)^2}$. This ensures that nearer neighbours get greater weight. Equation (2) is therefore the distance-weighted mean of the neighbours.

1.2.2 knn Regression

The knn predictions avoid the explicit approximation of the target function f in that they do not directly incorporate the features of the neighbours and the query structure in $\widehat{f}$. *Local Linear Regression* is another member of the family of knn predictions that does just that. It is defined as:

$$\widehat{f}(x) = b_1 a_1(x) + \ldots + b_n a_n(x), \tag{3}$$

i.e., a local linear model is learned, namely the coefficients $b_1 \dots b_n$, which form an explicit approximation to f over a local region surrounding x_q, namely the neighbours. To obtain the b_i, a system of linear equations has to be solved to minimize the error of $\widehat{f}$ on the data, one line for every neighbour:

$$\begin{aligned} f(x_1) &= b_1 a_1(x_1) + \ldots + b_n a_n(x_1) \\ &\vdots \\ f(x_k) &= b_1 a_1(x_k) + \ldots + b_n a_n(x_k) \end{aligned}$$

In order to retain the weighting between neighbours, weights $w_1 \dots w_k$ for the lines can be included in the calculation of the coefficients, which should again be a function of the distance to the query structure. This is called *Locally Weighted Regression*.

1.2.3 Properties of knn predictions

Knn predictions are lazy learning methods: for each distinct query a new approximation to the target function is created. This has important implications:

- The approximations are local and differ from one another; therefore, for the whole feature space, many different approximations are used at different locations. The single approximations are quite simple, but seen as a whole they can approximate a complex function. They are independent from one another and there is no need to model a very complex n-dimensional function explicitly. Local methods like knn or radial basis functions are also quite robust, because they are only dependent on the data points close to the query instance.
- A new model is built for every new query, i.e., generalization is deferred until a new query instance is to be predicted. In contrast to eager learning, the computational burden for lazy learning methods is quite high, since all the training is done at query time. Eager learning methods have a seperate training phase, in which a global model is learned from the whole database and then at query time the only work to be done is for the fixed model function to be evaluated on the query instance. So for lazy learning methods, it should be possible to

calculate the neighbours fast.

- Often, with lazy learning, all of the features are included in the model, which could be problematic, since in almost every case not all of the features are relevant. Especially when using structural information, there are substructures that are special cases of other substructures. Those are often highly correlated, because they probably trigger the same chemical reactivity, or they are not active at all. It is wise to incorporate only uncorrelated and relevant features into the model. Measures one can take towards excluding correlated features include finding out about the significance of features and then stretching the axis of the feature space accordingly, or using more elaborate methods like *Decision Trees* or *Principle Components Analysis*, which include only a subset of the features in the model or find the direction of greatest variance in the data.

- The weighting of neighbours at model-building time according to their distance from the query structure allows for using all training instances as neighbours instead of only the k nearest ones, i.e., if there are n training instances, set $k := n$. Doing so is no harm to model precision, because distant training points will have little effect on the approximation. The only obvious drawback is that model building takes longer.

We now turn to see, how knn techniques can be beneficial for Predictive Toxicology.

1.3 Predicting Quantitative Chemical Properties

As discussed in section 1.1, in Predictive Toxicology we have existing training data, containing structural, physicochemical or other information as well as measured toxicological activities of the compounds. We have one untested compound called the *query structure*, for which we intend to predict the toxicological activity.

1.3.1 (Q)SAR

(Quantitative) Structure-Activity Relationships is a model of a well defined biological process caused by a chemical compound, such as biological activity or chemical reactivity (herein collectively referred to as *activity*)[a], using chemical descriptors of the compound. (Q)SAR's most general mathematical form is:

$$\text{Activity} = f(\text{physicochemical and/or structural properties}) \qquad (4)$$

The target function f is to be learned from training data contained in *databases*, which store chemical compounds or structures along with some properties in quantitative or qualitative form. The learning can be done by different means, such as (multi)linear regression or neural networks. Common to all learning mechanisms is the effort to minimize the prediction error, i.e. the modelling funtion f is improved until some optimization criterion is reached.

According to [KH04], the task of extracting a (Q)SAR model from training data consists of the following steps:

1. Definition of the goal of the project and the purpose of the SAR models.
2. Creation or selection of the dataset.
3. Checking the dataset for mistakes and inconsistencies, and perform corrections.
4. Selection of the features relevant to the project and transformation of the data into a format which is readable by data mining programs.
5. Selection of the data mining technique.
6. Exploratory application and optimisation of the data mining tools to see if they provide useful results.
7. Application of the selected and optimized data mining technique to the data set.

[a]The "Q" in "(Q)SAR" refers to quantitative properties of a compound, not to the activity. Using qualitative properties only is known as "SAR"

8. Interpretation of the derived model and evaluation of its performance.

9. Application of the derived model, e.g., to predict the activity of untested compounds.

1.3.2 Training data requirements for (Q)SAR models

When dealing with (Q)SAR models, one must always be critical about conclusions drawn from the training data. There are often subtle problems embedded some of which will be elaborated in the following.

The training compounds should feature a large variety of activity values for the endpoint under investigation. If this is the case, then they are said to have a good *Predictive Capacity*. If the histogram of activity values shows a sparse distribution in the middle and a concentration at the ends of the range, then binning the data into "active" vs. "non-active" could be more appropriate. If the range spans several orders of magnitude, then log values should be used instead of the original values [CL04].

A common QSAR problem is the phenomenon of *Overfitting*, that is, the model is improved until it perfectly fits the training data, hampering the ability to predict unseen data. So there is a tradeoff between general predictivity and error minimization on the training data [TLL95]. This is especially likely to happen for nonlinear methods.

Bad external predictivity is often not only due to overfitting, but also due to the incorporation of structurally diverse and non-congeneric compounds, that act with different biological mechanisms, into the training data. On the other hand, if training data contains only compounds with similar structures and specific biochemical interaction then very meaningful models can be learned by QSAR methods. Unfortunately, existing databases used in the life sciences are from the first type rather than from the second type. Often, when predicting toxicological endpoints, the learning data is therefore preselected for a single *Mode of Action (MOA)*, i.e. the specific biochemical interaction through which a drug substance produces its pharmacological effect. If this is the case, non-structural descriptors such as the octanol-water partition coefficient (logP) can be used to learn very accurate models.

Sometimes, however, the training information in the databases is sim-

ply wrong, i.e., for some compounds the activity information differs from the real values. This happens often for very toxic compounds, where only small doses are needed to obtain a reaction, or from transcription errors. Compounds with wrong database information are called *outliers*. There is no way of excluding them from the training database beforehand. Therefore the accuracy of the model may be compromised.

The *Lazar* algorithm is designed to tackle the problems of non-congeneric training data and outliers, in that it uses structural information and avoids learning a global model for the whole training set. Rather, it uses k-nearest-neighbour techniques to predict a chemical structure on the basis of a structurally similar (congeneric) subset of the training set, namely the neighbours (see section 1.2). Since Lazar is an instance-based method, outliers will only have a local effect. In Lazar, the applicability domain of a training set is modelled explicitly, clearly indicating for every new prediction if it will be reliable. Inutitively, it is reliable if the neighbours are "similar enough" to base a prediction on them. The following section details those mechnisms.

2 Methods

This section first gives a high-level view of the Lazar algorithm. It then describes the discovery of relevant fragments. After that, the identification of neighbours to the query structure as well as model building and prediction using knn techniques is elaborated.

2.1 Lazar Classification System

The following introduces the big picture of the Lazar prediction system and describes the steps towards the integration of quantitative information.

2.1.1 Linear fragments

The idea is to apply machine learning and data mining techniques to non-congeneric data sets, i.e. compounds that do not share a common core structure. Chemical similarity in Lazar is therefore based on structural information with respect to some given endpoint, not on physicochemical properties. The structural information consists of linear fragments occuring in the chemical compounds. A linear fragment of a compound is a linear subgraph (or path) of the 2D graph representing the compound. Lazar uses a simplified version of MolFea, the Molecular Feature Miner [KDH01], to find all linear fragments of a compound, employing the SMILES representation language [JWD00].

Lazar infers toxicological activity of the query structure from its chemical structure, assuming that the biological activity of a compound is determined by its chemical structure (or some of its properties). More specifically, linear fragments are used for similarity measurements and prediction. Figure 2 shows an example structure. It consists of two carbon ring structures connected through an oxygen atom. An example linear fragment is marked.

Using linear fragments ensures a modest computational load. Preliminary

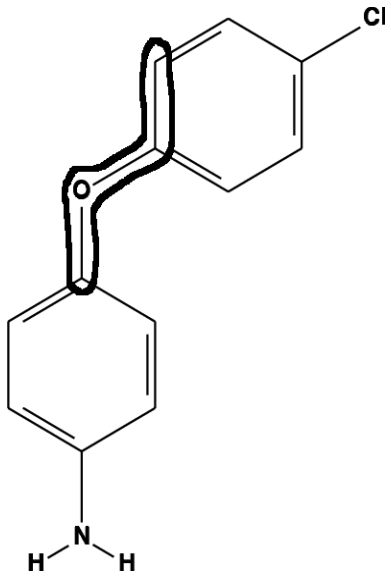


Figure 2: 2D representation of a compound ([KDH01], p. 2). It consists of two benzene rings connected through an oxygen atom. An example linear fragment is marked.

results show that using other types of features (e.g. subgraphs) yields no significant advantages.

A linear fragment f is a subfragment of a linear fragment f' if the 2D graph representing f is a subgraph of the graph representing f'. This induces an ordering $f \preceq_{sub} f'$. This ordering corresponds directly to specialization of fragments: f' is a specialization of f. A linear fragment f where there exists no other linear fragment f', such that $f \preceq_{sub} f'$, is called *most specialized*.

Example: We have a look at two linear fragments, given in the SMARTS language for describing molecular patterns [JWD00], along with the numbers of compounds, in which they occur in the EPAFHM database (this is an excerpt from the Lazar linfrag input file, see Appendix B):

f's structure, `c:n-C`, represents an aromatic `c` joined by an aromatic

```
f:    c:n-C       [ 18 83 311 423 437 515 537 572 ]
g:    c:n-C-C     [ 311 515 ]
```

bond to an aromatic `n` which is joined to an aliphatic `C` by an aliphatic (single) bond. Joining another aliphatic `C` by another aliphatic bond gives g. It holds that $f \preceq_{sub} g$. Therefore, g is a specialization of f. This can also be verified from looking at the compounds: f occurs in every compound that g occurs in, but not vice versa.

Most specialized linear fragments can be thought of as representing a whole cluster of linear fragments, namely their subfragments. The fragments contained in such a cluster are often highly correlated: they occur together, and it is possible that they are responsible for a single chemical mechanism if they are relevant for the current endpoint. A model based on linear fragments should try to take this into account.

2.1.2 Lazar workflow

A Lazar database consists of a set of pairs of chemical compounds and activity information for a specific endpoint. Given a database and a query structure Lazar gives a prediction about the activity of the query structure. The original algorithm for qualitative activity values [Hel06] works in four main steps, as depicted in figure 3.

In step 1., *all* linear fragments occuring in the training set are extracted. This is necessary, because ...

> "... especially with toxicological effects, we frequently face the problem, that biochemical mechanisms are diverse, purely understood or even unknown. It is therefore hard to guess (and select) the chemical features that are relevant for a particular effect [...]." [Hel04].

1. Extract all linear fragments from all structures in the training set (this can be done beforehand as this information never changes for a given database).
2. Find the linear fragments that occur significantly more frequently in active than in non-active training structures.
3. Find structures in the training set that are similar to the query structure with respect to the linear fragments found in the previous step (neighbours).
4. Predict the activity of the query structure based on the activity of the neighbours.

Figure 3: Main Lazar Workflow (quantitative)

In step 2., Data Mining is applied to select significant fragments. This automatic selection ensures that no important fragment can be missed. Step 3 consists of a weighted majority vote from the neighbours. Step 4 was originially conceptually and computationally simple and has since been extended to a more sophisticated procedure.

2.1.3 Integrating Quantitative Information

The original version works on a database of compounds with known qualitative activity values (*qualitative database*), i.e., for every compound in the database it is known whether it is active or inactive with respect to some clearly defined toxicological endpoint. This information is qualitative because it is a *yes/no* decision. The purpose of this work is to extend the algorithm to reliably predict quantitative values, i.e. *how active* a compound will be on a numerical scale.

The integration of quantitative training activities consists of changing the activity information about compounds in the database from qualitative to quantitative values (*quantitative database*). In step 2. of figure 3, a chi-square test is used to "identify fragments, that occur significantly more

frequent in toxic than in non-toxic compounds" [Hel06]. This test employs maximum-likelihood statistics obtained from counting the active and non-active compounds. With quantitative values the use of counting statistics is no longer possible, i.e. there is no maximum-likelihood value to use as expectation values for a potential chi-square test. Instead we are left with a range of unbinned activity values. To find out which fragments are part of specifically active compounds appropriate statistics are needed. This is covered in section 2.2.

The prediction process (step 4 in figure 3) must be adapted to the prediction of quantitative activity values. For qualitative values a weighted *Tanimoto Index* was used for prediction [Hel06]. For quantitative values no binary decision can be taken, and the activity values of the neighbours usually span several orders of magnitude. The prediction step is covered in section 2.4.

The four main steps of the algorithm are depicted again in the data flow diagram in figure 4, abstracted from the type of activity values. It applies therefore to both Lazar versions.

We have seen a general overview how Lazar uses linear fragments and instance-based learning to derive a local prediction. In the next section we get down to the details, explaining data mining techniques and different prediction models.

2.2 Significant Features

Kolmogorov-Smirnov-Test In [PTV93], a method for implementing the *Kolmogorov-Smirnov test (KS test)* is presented. Its purpose is, to find out whether a given set of data is drawn from a known probability distribution, for example from a normal distribution. It works for unbinned data with which we are dealing in our case of quantitative data. The widely

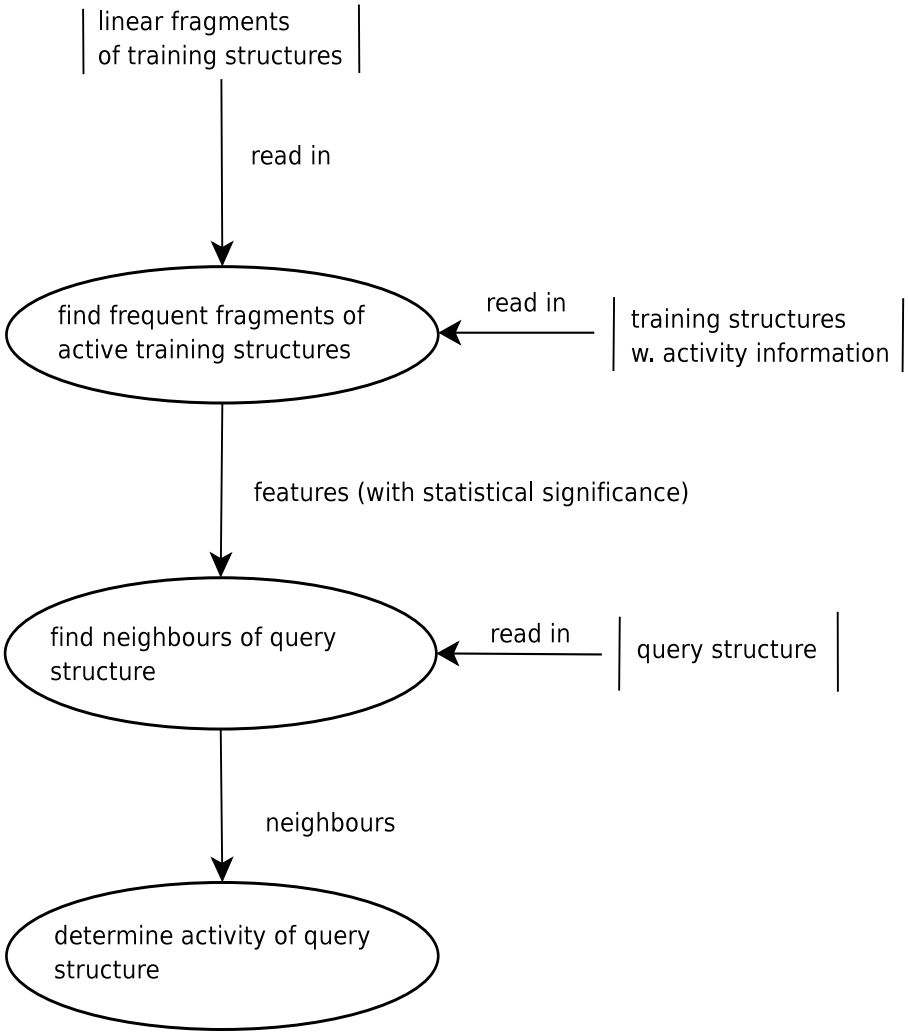


Figure 4: Data Flow Diagram of Lazar

used chi-square test works only for binned data and is not suitable for our purposes.

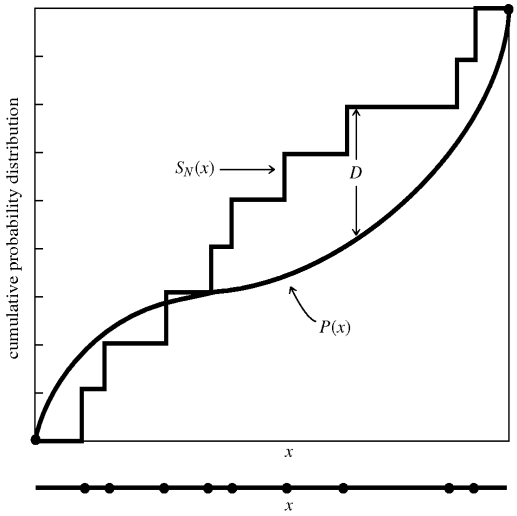


Figure 5: Kolmogorov-Smirnov test ([PTV93], p. 5): step size function $S_N(x)$, user defined probability distribution $P(x)$ and distance D.

The KS test compares a cumulative probability distribution $S_N(x)$ obtained from quantitative data to a user-defined cumulative probability distribution $P(x)$ (Figure 5). It measures the maximum distance D between $S_N(x)$ and $P(x)$. If D is "large" the test rejects the null hypothesis that both distributions are the same. The step size function is induced from an ascending ordered set $N = \{x_1, \ldots, x_n\}$ of quantitative values as follows: $S_N(x)$ is the fraction of data points to the left of a given value x. Obviously, between consecutive x_i's, this function is constant and jumps by the same constant $1/|N|$ at each x_i.

We want to know whether the activity values for compounds containing a specific fragment f differ significantly from those of all compounds. Thus we want to compare two sets of unbinned data, and we use two step size functions $S_{N_1}(x)$ and $S_{N_2}(x)$ instead of $S_N(x)$ and $P(x)$. N_1 contains the activity values of all training compounds containing f, and N_2, of all training compounds whether they contain f or not.

Definition 1 (Distance): The *distance* D between two step size functions is defined as

$$D = \max_{-\infty<x<\infty} |S_{N_1}(x) - S_{N_2}(x)| \tag{5}$$

D serves as input to the function *probks* together with the set sizes of N_1 and N_2:

$$p_f = probks(D, size(N_1), size(N_2)) \tag{6}$$

p_f indicates the probability that the two definition sets are not drawn from the same probability distribution function[b]. Therefore, it serves as an indicator for f significantly affecting the activity values for the endpoint under investigation. If $p_f > 0.95$ then f is called a *significant feature*.

Lazar employs an extended KS test which is equally sensitive for the whole range of activity values[c].

2.3 Neighbours

Neighbours to the query structure are determined by the p-weighted fraction of significant features shared with the query structure. Therefore, the neighbours are structurally similar compounds with regard to the current endpoint.

Definition 2 (Similarity): Let F be the set of significant features. For all compounds s_t, determine $sim(s_q, s_t)$, the *similarity* between query structure s_q and s_t[d]:

$$sim(s_q, s_t) = \frac{\sum_{f \in F} \{p_f^4 | f \subseteq s_q \wedge f \subseteq s_t\}}{\sum_{f \in F} \{p_f^4 | f \subseteq s_q \vee f \subseteq s_t\}} \tag{7}$$

All compounds s_t with $sim(s_q, s_t) > 0.3$ are considered neighbours of the query structure s_q. This threshold is introduced for efficiency reasons only.

[b] *probks* is defined in [PTV93]

[c] see also [PTV93]

[d] The exponent of 4 ensures that low values of p do not get too much weight.

One could use all training compounds as neighbours because of weighting in the prediction process (see section 1.2.3).

2.4 Predictions

This section introduces the application of k-nearest-neighbour strategies (see 1.2) in Lazar.

2.4.1 Median Model

The predicted activity is the *sim*-weighted median of the neighbours' activity values. The weighted median is similar to the weighted mean, the standard knn prediction, but more robust against extreme values.

Definition 3 (Weighted Median): Consider a set of n real numbers sorted in ascending order in a sequence $nr = \langle nr_1, \ldots, nr_n \rangle$. Every element nr_i of the sequence has an associated real value $w(nr_i)$ called its weight. Then the *Weighted Median Index* is defined as

$$idx = \frac{\sum_{i=1}^{n} i * w(nr_i)}{\sum_{i=1}^{n} w(nr_i)}. \tag{8}$$

The *Weighted Median* is obtained by interpolation between the values of the upper and lower element relative to the position given by the median index [e].

$$med = \frac{(nr_{\lfloor idx \rfloor} + nr_{\lceil idx \rceil})}{2} \tag{9}$$

Finer interpolations than taking the mean values of the two closest elements could of course be applied.

Example: Consider the ordered sequence $s = \langle 1, 3, 1, 2, 1, 1, 100 \rangle$ of

[e] $\lfloor\ \rfloor$ ($\lceil\ \rceil$) denotes rounding down (up) to whole numbers.

length 7. The mean value is $(1+3+1+2+1+1+100)/7 \approx 15.57$ whereas the median is $s_4 = 2$, because 4 is the index for the middle position[f].

The mean is greatly affected by the sequence member 100. This is not very sensible, as no number in the sequence is close to 15.57. The median, on the other hand, will always be reasonably close to other sequence members.

Definition 4 (Lazar Weighted Median): Consider the set of activities of the k nearest neighbours sorted in ascending order in a sequence $nb = \langle f(nb_1), \ldots, f(nb_k)\rangle$. Every $f(nb_i)$ has an associated value $sim(s_q, nb_i)$, consisting of the neighbour's similarity value to the query structure.

The *Lazar Weighted Median* is an application of the weighted median obtained by using the similarity values as weights:

$$\begin{aligned} idx &= \frac{\sum_{i=1}^{k} i * sim(s_q, nb_i)}{\sum_{i=1}^{k} sim(s_q, nb_i)} \\ \widehat{f}(s_q) &= \frac{(f(nb_{\lfloor idx \rfloor}) + f(nb_{\lceil idx \rceil}))}{2} \end{aligned} \tag{10}$$

2.4.2 Multilinear Model

In the multilinear model the activity is predicted by a weighted general linear model, based on the neighbours, implementing knn regression.

Definition 5 (General Linear Model): A *General Linear Model* is a vector c of coefficients that minimize the error on a system of linear equations,

$$y = Xc, \tag{11}$$

where y is a vector of n observations, and X is a k by m matrix of predictor variables.

[f] we use the unweighted median here, i.e., all weights are the same number unequal to zero

Definition 6 (Lazar Multilinear Model): Let nb_i be the i-th-nearest neighbour to the query structure s_q , let $act(nb_i)$ be the activity of nb_i, and let F be the set of features obtained from uniting the linear fragments of all neighbours with the linear fragments of the query structure:

$$F = (\bigcup_{i=1}^{k} features(nb_i)) \cup features(s_q) \tag{12}$$

Let f_j be the j-th most significant feature in F, and let w be a vector of weights for the observations. Then the Lazar multilinear model is a vector $c = (c_1, \ldots, c_m)$ that minimizes the error on a general linear model $y = Xc$ with weight vector $w = (w_1, \ldots, w_k)$ based on the k nearest neighbours to the query structure s_q and the m most significant features of the whole training database, where[g]

1. $y_i = act(nb_i)$, where nb_i is the i-nearest neighbour to s_q,
2. $X_{ij} = \begin{cases} p(f_j) & \text{, if } f_j \text{ occurs in } nb_i, \\ 0 & \text{, else,} \end{cases}$
3. $w_i = sim(s_q, nb_i)^4$.

The feature vector $x = (x_1, \ldots, x_m)$ of s_q is defined as:

$$x_j = \begin{cases} p(f_j) & \text{, if } f_j \text{ occurs in } s_q, \\ 0 & \text{, else.} \end{cases}$$

The prediction is obtained by evaluating x on the model c:

$$\widehat{f}(s_q) = c^T x \tag{13}$$

Each row in the $k \times m$ matrix X corresponds to a neighbour. The

[g]The exponent of 4 in the definition of the weights ensures that distant (unsimilar) neighbours don't get too much weight.

columns are indicators for linear fragments featuring the significance value for presence and zero for absence.

2.4.3 Cluster Model

In the following, a method for feature clustering that goes beyond the simple ordering of linear fragments according to their significance is elaborated.

Principal Components Analysis (PCA) is a powerful tool to identify patterns within data and express the data differently with the purpose of highlighting their similarities and differences. I will give an example-applied explanation of how PCA works, leaving out mathematical proofs. As it is a frequently applied technique it is well documented [Jol02].

Especially in high-dimensional data it can be difficult to find patterns which is also due to the fact that it is hard to visualize. If parts of the data are correlated, PCA decorrelates them and unites them to a set of new descriptors, the so-called principal components. PCA features also the possibility of compressing data, namely by omitting the most insignificant principal components.

In the multilinear model of Lazar there is m-dimensional data in the matrix X (see the previous section), one dimension for every descriptor (linear fragment). Every data point (compound) is therefore an tuple in the m-dimensional feature space. It is quite probable that many linear fragments have the same pattern of occurence and are responsable for triggering the same chemical mechanism (see section 2.1.1). If this is true, then many columns in X are highly correlated.

PCA proceeds in two steps: first the principal components of the data have to be calculated. They will form the unit vectors of the new descriptor space. In the second step they are aggregated in a so-called rotation

matrix which is then used to transform the data's coordinates into the new descriptor space.

There are different possibilities for computing the principal components. The common idea is to capture the direction of the largest variance in the data with the first principal component, the second largest variance with the second principal component and so on. The most insightful approach is to compute the principal components as the eigenvectors of the covariance matrix of the zero-centered data matrix D.

For n two-dimensional, real-valued and zero-centered (i.e. mean value subtracted) data points in the dimensions X and Y, one can compute the covariance between X and Y, $cov(X, Y)$ as:

$$cov(X, Y) = \frac{\sum_{i=1}^{n} X_i * Y_i}{n - 1}, \tag{14}$$

For more than two dimensions one can form a covariance matrix yielding the covariance between each pair of dimensions. The eigenvectors of this covariance matrix form the principal components. Ordering them in descending order by their associated eigenvalues yields a scaling of their importance: more important components have greater associated eigenvalues. Depending on the regularity in the data, i.e. the amount of correlation between the original dimensions, one can observe that summing up the eigenvalues in this ordering the sum of the eigenvalues is approached more or less quickly. With high correlation few principal components will be required to capture most of the variance.

Having determined all principle components one fills them in a so-called rotation matrix R columnwise from left to right in descending order of their importance. Performing matrix multiplication between rotation matrix and data matrix transforms the data into the new descriptor space: the transformed data D_{trans} can be computed as: $D_{trans}^{T} = R^{T} * D^{T}$.

Example: The application of PCA can be understood most easily in two dimensions. The example in figure 6 is due to Smith [Smi02].

DATA:		ROTATION MATRIX:		TRANSFORMED DATA:	
2.5	2.4	-0.677873	0.735179	-0.82797	0.175115
0.5	0.7	-0.735179	-0.677873	1.77758	-0.142857
2.2	2.9			-0.992197	-0.384375
1.9	2.2			-0.27421	-0.130417
3.1	3			-1.6758	0.209498
2.3	2.7			-0.912949	-0.175282
2	1.6			0.0991095	0.349825
1	1.1			1.14457	-0.0464173
1.5	1.6			0.438046	-0.0177646
1.1	0.9			1.22382	0.162675

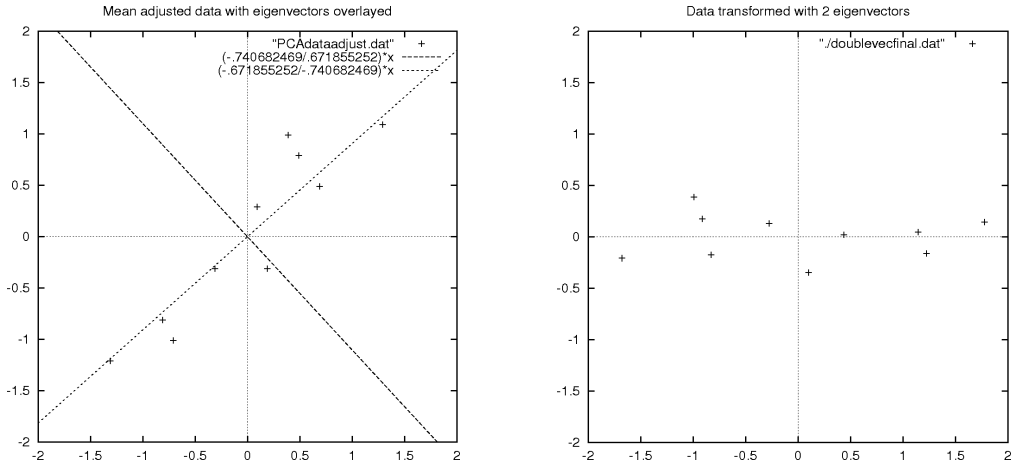


Figure 6: **Top:** Original data (left), rotation matrix calculated by PCA consisting of the eigenvectors of the covariance matrix of the zero-centered data (middle) and the transformed data obtained by multiplying the data with the rotation matrix (right). **Bottom:** Data plot of zero-centered data overlayed by the principle component vectors (left) and transformed data plot (right).

All key concepts of PCA can be observed from this example:

- The data has a strong *pattern*, i.e. the two dimensions are strongly correlated: the data points form a "cloud" around the line $y = x$ as can easily be seen from the left plot.

- The *eigenvectors* of the covariance matrix are plotted over the data as dashed lines. They form the columns of the rotation matrix. The first eigenvector, $\binom{-0.677873}{-0.735179}$, points roughly into the direction $y = x$ and has the largest associated eigenvalue. It captures the main trend in the data like a linear best fit. The second eigenvector, $\binom{0.735179}{-0.677873}$, captures the secondary, less important, trend, namely that all points are off to the side to some amount. The corresponding eigenvalues are 1.28403 and 0.0490834, respectively. This means that the first eigenvector captures over 96% of the data's variance (the sum of the eigenvalues).

- The two eigenvectors are the principal components that give the directions of the new coordinate system. The right plot depicts the data after transformation in the new coordinate system. This gives us the very same data, only in the terms of it's principal components: it is now expressed in terms of pattern between the data. Another way of saying it is: we have rotated our coordinate system 45 degrees to the left because the main information lies in a 45 degree angle to the left. Of course this will get more complex in higher dimensions, but the principle is the same.

- It is possible to reduce the amount of data by losing some of the information: one can omit some of the less important principal components. In our example one could omit the second component which would simply eliminate all information in the transformed data concerning the second dimension. This would have the effect of projecting all points on the x-axis, losing the y-information. Eliminating the eigenvectors with the lowest eigenvalues is optimal in the sense that the data cannot be shrinked without losing less information.

Lazar uses PCA in the cluster model to compress its structural data. Starting with the data matrix X from the multilinear model, X is enlarged by

one more row: the query structure's data, contained in the vector x in the multilinear model, is also incorporated into X for transformation. After transformation of X to X_{trans} using PCA the extra row is removed again from X_{trans} and stored in the new transformed feature vector x_{trans}. Then the very same regression is applied as in the multilinear model, but using X_{trans} instead of X and x_{trans} instead of x. The ideas behind that are:

- Co-occurring linear fragments are clustered together: Assuming that most specialized linear fragments represent a whole family, or cluster, of linear fragments, they are able to describe the data in a more concise and human-accessible form. Furthermore, if linear fragments are significant for triggering certain chemical mechanisms it is easier to conclude a mechanical explanation for the toxic effect with clustering.

- Enabling the use of more linear fragments while reducing data size: In the case that quite a lot linear fragments are strongly correlated, PCA will be able to cover most of the data's variance with only few descriptors. While the ordinary multilinear model was restricted to a maximum of 50 features due to efficiency reasons it is possible to now use 500 or more features with PCA: by leaving out the most insignificant principal components, covering "only" 99% of the data's variance, the data is reduced to an average descriptor space of 10 to 30 principal components.

In this section we learned how Lazar derives quantitative predictions using automatically mined significances of linear fragments on the lowest level. Building on those significances neighbours are detected and used in three different models. In the following we will investigate the predictivity of the different models and also compare them with the results of other approaches.

3 Results

This section gives details about the predictivity of quantitative Lazar predictions and relates them to results of others.

3.1 Crossvalidation

Two databases were used for the validation step, drawn from U.S. EPA's DSSTOX database [EPA], namely the EPA Fathead Minnow Aquatic Toxicity Database (EPAFHM) and the FDA Center for Drug Evaluation & Research - Maximum (Recommended) Daily Dose Database (FDAMDD), specifically versions v3b_617_10Apr2006 and v2b_1217_10Apr2006, respectively. EPAFHM and FDAMDD are among the most popular public databases for validation purposes.

For validation of Lazar *Leave-One-Out Crossvalidation* (LOO) was used. With LOO, every single compound in the database is predicted once on the basis of the rest of the compounds in the database. Put it another way, for a database containing n compounds a prediction for an external test set containing exactly one compound is performed n times, each time using the remaining $n-1$ compounds as training instances. Lazar takes care that for every single prediction the significance of features of training compounds is calculated from scratch achieving an unbiased prediction[h].

However, there are several reasons why the number of predictions made could differ from the number of instances in the databases and could also vary from model to model.

- No neighbours to the query structure could be identified.
- Some compounds are contained multiple times in the databases. Lazar

[h]This is an important point in LOO, which many implementations take not enough care of.

detects multiple instances by creating canonical descriptions of all compounds and removes them from the database prior to the prediction process.

- For the multilinear and the cluster model goodness-of-fit parameters were made available by the regression component. Those were used to exclude cases where no acceptable fit could be achieved by the model.

3.2 Perfomance and Applicability Domain

For LOO, performance was measured with R^2, the coefficient of determination.

Definition 7: the *Regression Sum of Squares* is defined as

$$RSS = \sum_i (pred_i - db_i)^2, \tag{15}$$

and the *Total Sum of Squares* is defined as

$$TSS = \sum_i (pred_i - \bar{db})^2, \tag{16}$$

where i ranges over all compounds, $pred_i$ is the Lazar prediction for compound i, db_i is the database activity for compound i, and $\bar{db}$ is the sample mean of all database activities.

Then, R^2 is defined as

$$R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}}. \tag{17}$$

It is also called the proportion of the variance explained by the model. R^2 is one of the most common performance indeces in the field of predictive toxicology. In the context of LOO R^2 is also called Q^2_{LOO}. In the context of predictions made by Lazar, R^2 is always the same as Q^2_{LOO}.

Predictive performance in Lazar is closely related to how similar the neighbours are to the query structure. The best performance is obtained for query compounds with a lot of very similar database structures. If this is true for a query structure, then it is said to fall within the applicability domain of the database. Conversely, if the neighbours only show a low similarity, no meaningful prediction can be obtained for that structure.

Definition 8 (Lazar Confidence Index): For a prediction for structure s_q, the *Lazar Confidence Index* $conf(s_q)$ is the sample mean of the similarities of the neighbours s_i used for the prediction [i]:

$$conf(s_q) = \frac{1}{n} \sum_{i=1}^{n} sim(s_i, s_q)^4. \tag{18}$$

Definition 9 (Lazar Applicability Domain): The *Lazar Applicability Domain* of a database D is a real number $ad(D)$, where $0 \leq ad(D) \leq 1$.
A structure s_q falls within the applicability domain of D if $conf(s_q) \geq ad(D)$.

The applicability domain varies between different endpoints, because different endpoints correspond to different functional mechanisms that require more or less chemical similarity. Different values for the applicability domain in conjunction with LOO runs can help figuring out the value that is most suitable for the current pair of database and endpoint to obtain meaningful predictions.

Figure 7 summarizes Lazar prediction performance for the multilinear and median model, figure 8 for the cluster model. The 20 most significant features were used for regression in the multilinear model. In cases where less than 20 neighbours could be identified, the number of features was identical to the number of neighbours (the number of neighbours corresponds to

[i] The exponent of 4 ensures that distant (unsimilar) neighbours don't get too much weight.

the number of rows and the number of features corresponds to the number of columns in X). In the cluster model the 300 most significant features were used. With PCA this data was compressed to mostly 5-20 principal components (the information loss due to compression was limited to a maximum of 7.5% of the data's variance). In cases where less neighbours than principle components could be found the rows in X were inserted multiple times to be able to use all principle components in the following multilinear regression.

D (# predictions)	**ad(D)**	**Model**	**#(ad(D))**	$\mathbf{Q^2_{LOO}}$
FDAMDD (1217)	0.8	multilinear	23	83%
		median	23	82%
	0.4	multilinear	89	78%
		median	93	74%
	0.2	multilinear	234	72%
		median	238	66%
	0.08	multilinear	522	70%
		median	544	60%
	0.0	multilinear	948	53%
		median	1146	37%
EPAFHM (580)	0.3	multilinear	13	81%
		median	12	80%
	0.2	multilinear	34	73%
		median	32	72%
	0.1	multilinear	140	72%
		median	141	71%
	0.0	multilinear	574	36%
		median	574	44%

Figure 7: Quantitative Lazar Performance for leave-one-out crossvalidation using multilinear and median model. The first two columns give training dataset and size along with different applicability domains. The following columns give the performance results for the two different models.

Figures 9 and 10 show the plotted results for FDAMDD and for EPAFHM, both using the multilinear model with applicability domains of 0.2 and 0.1, respectively. The complete plots for all runs corresponding to figures 7 and 8 can be found in Appendix C.

D (# predictions)	ad(D)	#(ad(D))	Q^2_{LOO}
FDAMDD (1217)	0.8	17	-4%
	0.4	78	46%
	0.2	211	52%
	0.08	478	54%
	0.0	969	34%
EPAFHM (580)	0.3	6	1%
	0.2	19	55%
	0.1	119	53%
	0.0	320	25%

Figure 8: Quantitative Lazar Performance for leave-one-out crossvalidation using the cluster model. The first two columns give training dataset and size along with different applicability domains. The following columns give the performance results for the model.

A plotted result always consists of a pair of plots: The *accuracy plot* (upper parts of figures 9 and 10) and the *prediction plot* (lower parts of figures 9 and 10). The accuracy plots show R^2 vs. confidence in a cumulative fashion: The first data point to the left (with the highest confidence) corresponds to the (single) prediction with the highest confidence, the second data point to the two predictions with the highest confidences, and so on. The prediction plots show predicted vs. database activities ($-log$ values): The light-gray compounds do not fall into the applicability domain, while dark-gray and black dots do. Black corresponds to the best confidence values, dark grey to confidence values within the applicability domain. A linear best fit line for the compounds in the applicability domain is drawn on top of the plot. Obviously, it should lie closely to $y = x$, if there is no systematic error in the predictions.

3.3 Predictivity

The multilinear model is superior to the median model in nearly all combinations of database and applicability domain. This can be easily seen from figure 7. The two models show a parallel behavior in the accuracy

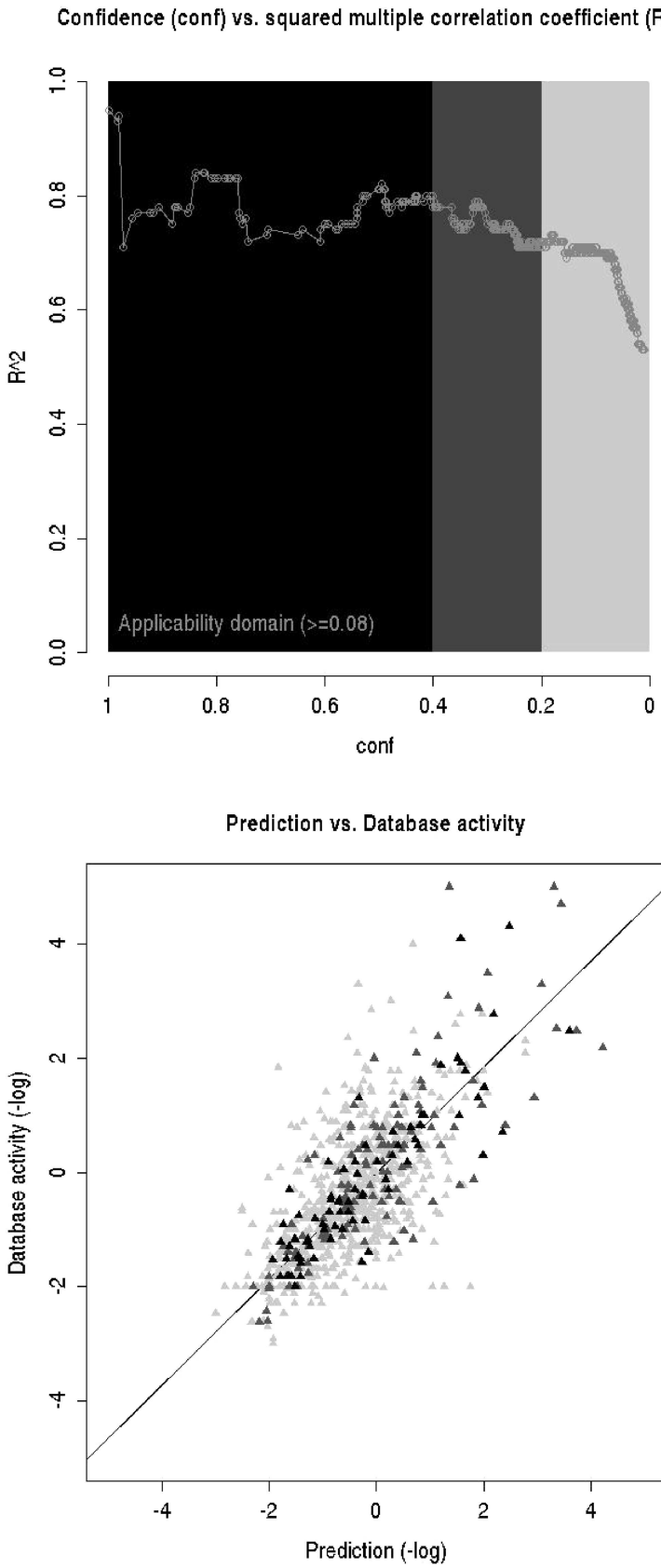


Figure 9: Quantitative Lazar performance for FDAMDD, applicability domain set to 0.2, incorporating 234 compounds (multilinear model). **Upper:** Confidence vs. cumulative R^2. **Lower:** Scatterplot od predicted vs. database activity.

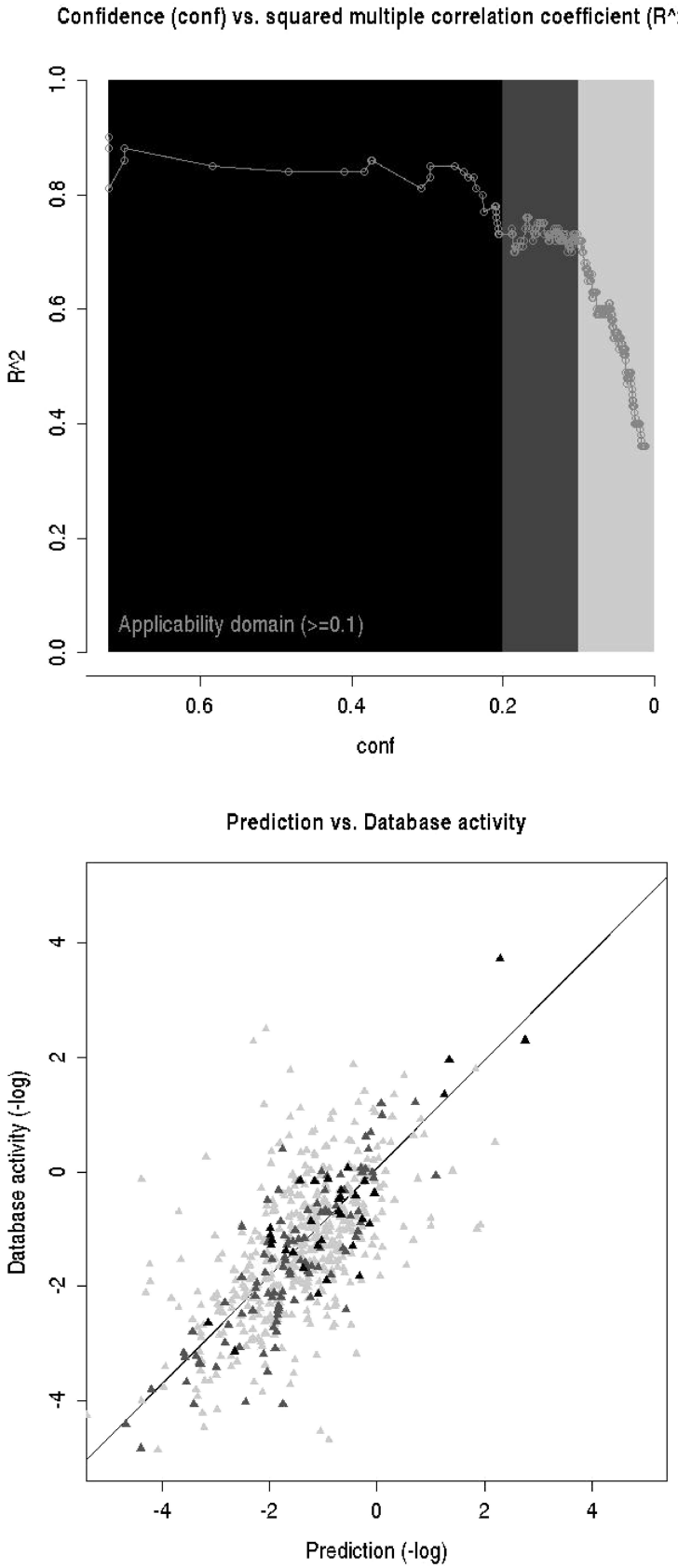


Figure 10: Quantitative Lazar performance for EPAFHM, applicability domain set to 0.1, incorporating 140 compounds (multilinear model). **Upper:** Confidence vs. cumulative R^2. **Lower:** Scatterplot od predicted vs. database activity.

plots, behaving nearly monotonous[j]. The cluster model fails to achieve accuracy rates similar to the other two models. It is very unstable in the beginning and does not exceed R^2 rates of 60%. Even when reaching this maximum (FDAMDD with applicability domain 0.08), the best-fit line in the corresponding plot (see Appendix C) reveals a systematic error in that the cluster model overestimates activities.

The mean similarity of the neighbours is used as a confidence index of how reliable the predictions will be. With the multilinear model, this index is obviously meaningful: with decreasing confidence, the predictions get less accurate. The accuracy plots of figures 9 and 10 show that the Lazar confidence index (see section 3.2) is quite a good estimator for prediction quality. High confidence values are given most often to good predictions while they are rare together with low confidence values. This is clearly indicated by the cumulative R^2 values and confidence is therefore usable as a measure for the applicablity domain of a database.

In the case of the databases used here the most sensible values are probably 0.08 for FDAMDD and 0.1 for EPAFHM. Using these values puts 522, respectively 140, compounds in the applicability domain.

3.4 Comparison to QSAR models

The following results for Fathead Minnow data were achieved using global (Q)SAR models. For each result we will summarize the compounds and features used, and the training and test set settings as given in the articles. Short comments from the author of this paper are printed in *italics* at the end of each summary.

[j]The "jumpy" behavior of the cumulative R^2 line at the beginning is the consequence of too few predictions contributing values. As can be seen, this stabilizes with more data points.

T. Öberg, 2004 [Ö04], Partial Least Squares Regression Öberg payed special attention to the applicability domain. The 611 compounds were closely investigated prior to model generation, and only 311 "were selected on the basis of classification of their modes of action as narcosis (narcotics modes I-III)"[k]. Those were then split into a training set of 208, and an external test set of 103 compounds.

For inclusion in the final model 218 descriptor variables were selected based on significance tests. This also resulted in an exclusion of nine further compounds from the training set. Subsequent Principal Component Analysis [CL04] yielded five latent variables covering 69.9% of the descriptor variables' variance. *Partial Least Squares Regression* method was used for modeling.

Öberg reports a Q^2 for leave-one-out crossvalidation for the training set of 87.6% and 88.8% for the test set. He compares his results to a simpler linear ANOVA model, which yielded $R^2_{cal} = 60.3\%$.

The manual preselection of compounds by a human expert according to mode of action is still common using QSAR models. As a global model without distance weighting is used the training compounds and the query structure must be congeneric. However, this is error prone (see section 1.3). Lazar automatically mines congeneric compounds.

E. Papa et.al. [PVG05], Multilinear Regression A set of quantitative features obtained from the data were used for Multiple Linear Regression. The features were calculated using the DRAGON software, including 1D, 2D and 3D information. The set was reduced by genetic algorithm subset selection. The compounds were classified a priori into narcotics, polar-narcotics, reactive or specific acting compounds (MOA1 - MOA4, respectively). Special attention was given to the logP feature which describes

[k]These modes correspond to LC_{50} values of < 100 mg L^{-1}, meaning acutely toxic.

the octanol-water partition coefficient. Different types of logP were used for modeling: AlogP, MlogP and ClogP (only ClogP values are reported in figure 11).

The authors report Q^2_{LOO} values for the 3x4 setting logP-type vs. MOA-type. MOA1-based classification yields much higher Q^2_{LOO} rates, up to 92%, compared to classification based on all compounds.

Preselection according to mode of action (different narcotics) is done again. Additionally, logP is used as a feature which is a physicochemical descriptor known to exhibit a nearly linear relation to narcotic modes. Lazar uses only structural information.

D.V. Eldred et.al. [EWJK99], Multilinear Regression, Computational Neural Network (CNN), Genetic Algorithm The authors used 287 compounds as training set, 88 compounds were divided in two groups for crossvalidation and prediction in the case of CNN, or both for external prediction in the case of multilinear regression. 242 topological descriptors were generated from connection table information yielding only structural properties. They were reduced to 123 by objective feature selection. Simulated annealing was used for the multilinear model.

The authors report a correlation value for the prediction set in the multilinear case: $R = 0.86$ ($R^2 = 0.74$). The other methods show better RMSE values, but no R values are reported.

The multilinear model is similar to Lazar. It uses only structural properties computed from the compounds.

S.P. Nicolescu et.al. [NAHL04], Probabilistic Neural Network A probabilistic neural network was trained for 886 compounds obtained from EPA's AQUIRE database. They were randomly split into 800 training and 86 test compounds. The features consisted of structural information, i.e.

the number of occurences of certain groups similar to [MY01]. Two models werde learned based on a group contribution method of which the better one is listed in figure 11.

Considering the fact that no preselection has been done this is an impressive result. However, neural networks suffer frequently from overfitting and a lot of manual tweaking can be done here.

M. Pavan et.al. [PNW06], Multilinear Regression A set of quantitative features obtained from heterogenous data was used for multilinear regression. Similar to [PVG05], the features were calculated using the DRAGON software, included 1D, 2D and 3D information and were reduced by genetic algorithm subset selection. The model was externally validated against 57 chemicals. The authors report $Q^2_{LOO} = 80.1\%$ and $Q^2_{ext} = 72.1\%$.

Again a good result, using quantitative features. Quantitative features are a valuable feature and could be part of a future release of Lazar.

Summary Figure 11 shows the QSAR results in compact form. Q^2_{LOO} is defined as R^2 obtained through leave-one-out crossvalidation, and Q^2_{ext} is defined as R^2 on an external test set.

Method	Q^2_{LOO}	Q^2_{ext}	R	FT	LS
PLSR [Ö04]	88% (208)	89% (103)	-	phys-chem.	moa
MLR [PVG05]	67% (469)	-	-	topol.	moa
MLR [EWJK99]	-	-	86% (88)	struct.	all
NN [NAHL04]	-	78% (86)	-	struct.	all
MLR [PNW06]	80% (408)	72% (57)	-	topol.	all

Figure 11: **Comparison of Results.** The numbers in round brackets indicate the number of predicted compounds. **Shortcuts:** PLSR: Partial Least Squares Regression, MLR: Multilinear Regression, NN: Neural Network, FT: feature type, LS: learning/prediction set composition ['moa': preselected compounds according to their mode of action; 'all': no preselection]

For the EPAFHM database we have seen the work of others trying to accomplish the task to predict activity using global models. The approaches show a variety of techniques for model building and feature selection: multilinear regression is a common approach, eventually combined with preselection according to mode of action, but also computational neural networks, which are able to model arbitrary functions, but suffer easily from overfitting. Many different types of features were used, reaching from computed structural and physical properties to physicochemical descriptors drawn from literature.

The purpose of this section was to report the predictive performance of Lazar on two different, frequently used databases and the performance of others trying to accomplish prediction by different (Q)SAR models. The next section addresses conceptual aspects, discusses predictive performance and highlights some crucial points in the Lazar implementation that are important for runtime performance and precision.

4 Discussion

This section adresses important conceptual issues, discusses predictive performance and explains technical implementation details.

4.1 Conceptual Aspects

Lazar is a hybrid conception of both lazy learning and (Q)SAR methods. It is an instance-based method because it takes the query instance and its neighbours, weighted by similarity, into account when building a model. It also employs classical SAR methods by doing multilinear regression on qualitative data, namely the (weighted) presence or absence of fragments.

Lazar takes into account the important issues about lazy learning methods (see section 1.2):

- Lazar supports the extraction of all linear fragments contained in the database beforehand so that their occurences are known when the program starts. This enables a fast calculation of the fragments' significance values and of the neighbours.
- The more fragments a compound shares with the query structure the greater the similarity will be between the two (if they are the same it will be 1.0). The less fragments they share the smaller the similarity will be (if they do not share any features it will be 0.0). This is (inversely) related to euclidean distance.
- Only the most significant features are used to determine similarity. In the cluster model they are also uncorrelated. The accuracy plots for the median and multilinear model show that similarity is a meaningful confidence index. It can be assumed that it is also a meaningful distance measure.

- Weighting by similarity allows for using in principle all training compounds for each prediction. By calculating the weighted median, or weighting each row in X in the multilinear and cluster model, similarity is taken into account in all models.

- The ordering of features satisfies the constraint about using relevant information. Only the most significant features are used for supplying X with values. In the cluster model the features are also decorrelated.

4.2 Predictive Performance

The comparison of Lazar performance to QSAR approaches for the Fathead Minnow database (see section 3.4) show that Lazar ranks with recent models. R^2 results between 70% and 80% in the different applicability domains put it on a competitive level, especially with FDAMDD, where approximately half of the compounds could be classified with R^2 rates of about 70%.

One would have expected for the cluster model to achieve accuracy rates at least comparable to the multilinear model, since it is in principle superior to the latter, employing a much more meaningful feature space based on the principal components of the data. Additionally, it uses ten times the amount of features incorporating much more potentially meaningful descriptors. Additional runs using 500 or more features and/or a compression loss of less than 2.5% yielded no substantial increase of performance. This suboptimal behavior may be related to the following findings:

- In the matrix X of the multilinear model, each entry is basically a qualitative entry, i.e., it states whether the fragment is present in the compound or not although it is weighted by significance (see section 2.4.2). Normally, with PCA, one would expect to have real quantitative entries.

- The feature space (fragments) is very high-dimensional compared to the number of cases (neighbours), which make up the rows in X. Often the number of fragments is multiple times the number of neighbours, which causes a sparsity in the input data for the PCA.

- In the multilinear model, the locally weighted regression uses only the 20 most significant fragments out of usually several hundreds (we are looking at all fragments contained in any of the neighbours and the query structure). As a consequence X features many zeroes indicating a missing fragment. For multilinear approximation the many zeroes are an advantage, effectively reducing the number of dimensions. When using PCA in the cluster model X contains several hundred fragments. Although PCA transforms X to a new descriptor matrix X_{trans} with much less dimensions, the zeroes aren't retained.

- As can be seen from figure 8 quite a few compounds had to be excluded from the LOO run. This happened because the multilinear approximation to the PCA-transformed data was too bad, yielding standard errors of the fit above a threshold of 2.0. This didn't happen for the multilinear model.

It seems that for several quite subtle reasons PCA damages linear relations and that it can't make up for that flaw with the advantage of a broader feature space. It rather seems that the presence or absence of the 20 most significant fragments (which may be correlated with each other) suffice to make up quite a good feature space for multilinear regression.

4.3 Technical Implementation

Here we inspect several important implementational aspects of model building and prediction concerning mainly numerical issues.

All models strip redundant compounds from the training set. It is possible that the same compound occurs repeatedly in the database as the SMILES notation is not canonical. Therefore, duplicate compounds are removed from the training set before the determination of neighbours begins. To this end Lazar uses OpenBabel functionality to convert SMILES code into InChi format which is intended to be canonical [InC].

4.3.1 Multilinear Model

Once the neighbours have been determined, weighted linear regression is performed to obtain a model. We compute the best-fit parameters c of the model $y = Xc$ for the observations y with weights w and the matrix of predictor variables X as declared in section 2.4. The variance-covariance matrix of the model parameters is estimated from the scatter of the observations about the best-fit. Lazar returns the sum of squares of the residuals per degrees of freedom (χ^2/df) from the best-fit and prints them out (see Appendix B).

For a prediction not necessarily all compounds count as neighbours: there is a cutoff due to efficiency reasons at a similarity value of 0.3 (see section 2.3). In most cases, the number of neighbours ranges between 20 and 60, but Lazar makes a prediction even if there is only one neighbour to the query structure. This does not necessarily give a poor prediction.

The best-fit is found by singular value decomposition of the matrix X using a modified Golub-Reinsch singular value decomposition (see also *Cluster Model*) algorithm with column scaling to improve the accuracy of the singular values. Any components which have zero singular value (to machine precision) are discarded from the fit. For numerical reasons it was necessary to limit the number of fragments (columns) to the number of rows (neighbours) in X, because the regression component demands at least as many rows as columns. Therefore, there were also predictions made with

less than 20 fragments.

4.3.2 Cluster Model

In the cluster model, there are different ways of determining the principal components of the data. Lazar uses an approach that is well known from linear algebra.

Suppose M is an m-by-n matrix whose entries are real numbers. Then there exists a factorization of the form

$$M = U\Sigma V^*, \tag{19}$$

where U is an m-by-m unitary matrix, the matrix Σ is m-by-n with nonnegative numbers on the diagonal and zeros off the diagonal, and V* denotes the conjugate transpose of V, an n-by-n unitary matrix. Such a factorization is called a *Singular-Value Decomposition (SVD)* of M. Typical SVD implementations such as the ones used in Lazar have quadratic computational complexity.

The singular value decomposition is very general in the sense that it can be applied to any m-by-n matrix. SVD is related to eigenvalue decomposition: in the special case, that M is a Hermitian matrix which is positive semi-definite, i.e., all its eigenvalues are real and non-negative its singular values and singular vectors coincide with its eigenvalues and eigenvectors and can be found out by the above factorization.

The calculation is done by a singular value decomposition of the zero-centered data matrix, not by using the covariance matrix. This is generally the preferred method for numerical accuracy. The main functions used are the LAPACK routines of the *Fortran* language, DGESDD and ZGESVD.

4.3.3 Computational Complexity

Lazar is essentially a step-by-step procedure (see figure 4). Assuming that linear fragments have been extracted beforehand, i.e., the occurences of linear fragments in database compounds are known, the prediction process for a given query instance consists of three main steps: determining significant features, determining neighbours and finally computing a prediction value. The first two steps are covered by the UML diagram in Appendix D.

1. Determining significant features consists of gathering together the step-size function data for the KS-test, requiring linear time to extract all activity values and quadratic time to extract activity values for all compounds that contain the respective fragment. The KS-test itself requires linear time and some constant operations for all fragments.

2. Determining neighbours consists of calculating similarity values for all training compounds and selecting those with similarity greater than 0.3. This is implemented as to successive loops, each running in linear time.

3. Before the actual prediction starts, Lazar sorts neighbours and/or significant features using library routines in quadratic time. The complexity for the actual prediction depends on the model used. Calculating the median requires only linear time. Multilinear regression takes quadratic time using SVD, as does PCA. Therefore, the median is of linear, multilinear and cluster model are of quadratic complexity.

In summary, Lazar is of quadratic complexity in the size of the training set ($O(n^2)$).

The next section walks through an example prediction for a specific compound showing neighbours and their similarity values.

5 Practical Applicability: An Example

In this section the application of Lazar is demonstrated in a transparent way for an example compound. Lazar fulfills the requirements for (Q)SAR models as demanded by EU's REACH legislation.

Example: We have a look at a specific prediction from the FDAMDD database LOO run performed with the multilinear model. Lazar identifies 15 neighbours for the query structure `O=C1NC(=O)NC(=O)C1(CC=C)CC=C`. Some of its neighbours are depicted in figure 12. Figure 13 shows the structure graphs.

#	SMILES	Similarity	Activity
1	`N1C(=O)NC(=O)C(CC=C)(C(C)C)C1=O`	1.0	0.426511
2	`C1(=O)N(C)C(=O)NC(=O)C1(C(C)C#CCC)CC=C`	0.902632	0.30103
3	`C1(=O)C(CC=C)(CC(C)C)C(=O)NC(=O)N1`	0.878689	0.69897
5	`C1(=O)C(CC)(CC)C(=O)NC(=O)N1`	0.730738	1.0
8	`C1(=O)C(C(C)CC)(CC)C(=O)NC(=O)N1`	0.655752	0.30103
9	`C2(=O)NC(=O)C(C1=CCCCC1)(CC)C(=O)N2`	0.468516	0.824126

Figure 12: Some neighbours to the query structure, along with their similarity and activity values

The neighbours share a common core structure, namely the ring structure, and up to three double bonds with oxygen atoms. With decreasing similarity other fragments occur, attached to the ring, added to or replacing other fragments.

All linear fragments have a significance value attached indicating the importance towards triggering the endpoint, in this case the recommended maximum daily dose of a chemical (log values). Lazar uses the linear fragments, activities and similarities of the query structure and/or the neighbours to obtain a weighted multilinear model as described in section 2.4.2.

Figure 14 shows the prediction results. Lazar outputs the χ^2 value and the standard error of the fit. Those goodness-of-fit indicators tell us how

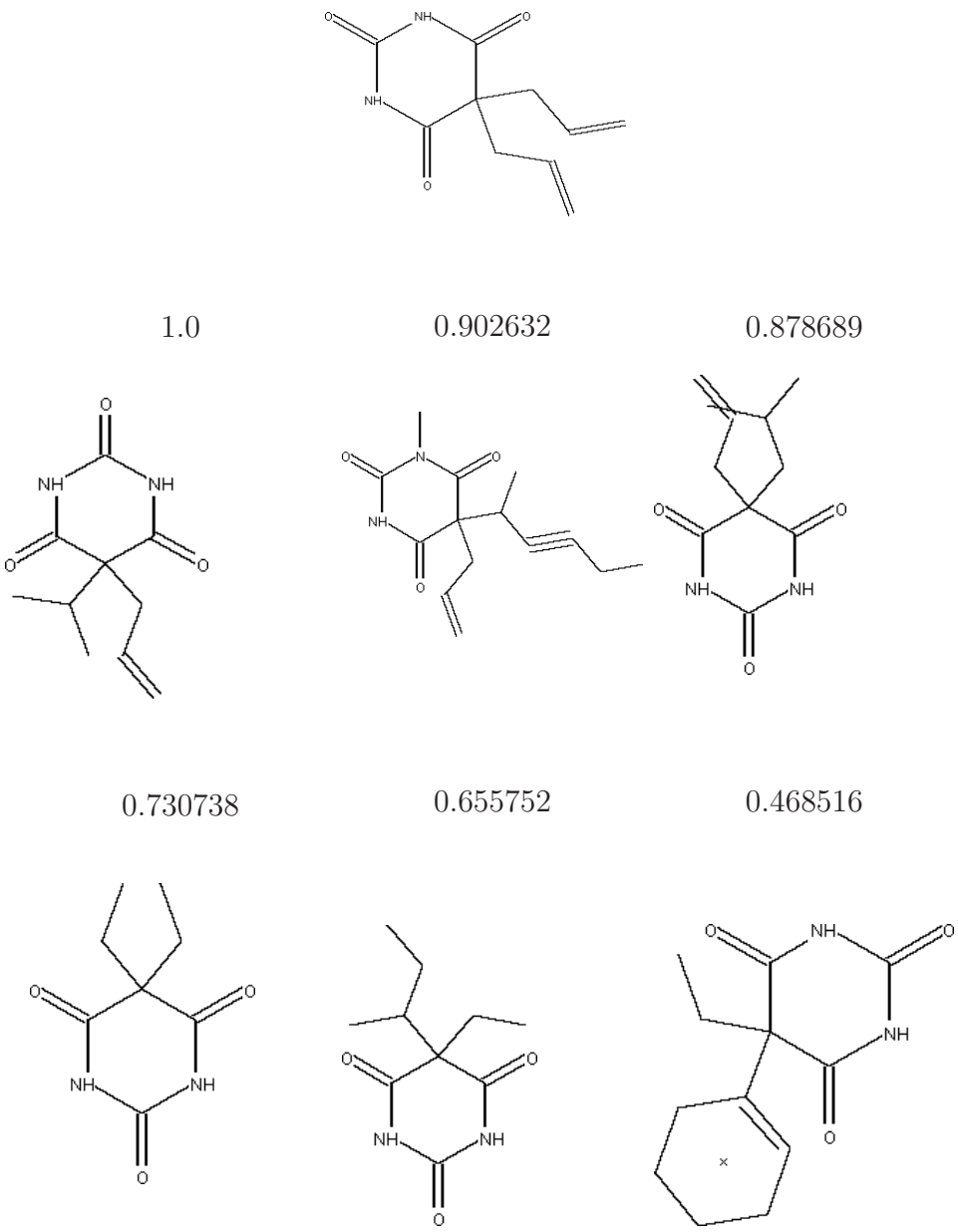


Figure 13: Some neighbours to the query structure (top), along with their similarity values. The depicted compounds are arbitrarily rotated (rotation is not encoded in the SMILES standard).

well the data could be fitted by the multilinear model. In our example the fit is quite good showing a low χ^2 value. Indeed the prediction fits the database activity quite well as indicated by the confidence. It is much greater than the applicability domain of 0.08 for this data set (the query structure falls within the applicability domain).

χ^2	0.0183677
standard error of the fit	0.449387
prediction	0.532071
confidence	0.284365
database activity	0.522444

Figure 14: Prediction results for the query structure

The Lazar algorithm provides therefore an unambiguous algorithm together with transparent information how the prediction was achieved. Chemical experts can use the neighbours to identify or confirm substructures relevant for the endpoint under investigation. We shall now see, how this corresponds to the requirements for (Q)SAR models, put forth by the European Union.

On december 18, 2006, the European Union (EU) decided to establish a regulation "concerning the Registration, Evaluation, Authorisation and Restriction of Chemicals (REACH)" which was put into force june 1, 2007 [REA06]:

> The REACH regulation gives greater responsibility to industry to manage the risks from chemicals and to provide safety information on the substances. Manufacturers and importers will be required to gather information on the properties of their substances, which will help them manage them safely, and to register the information in a central database. A Chemicals Agency will act as the central point in the REACH system: it will run the databases necessary to operate the system, co-

> ordinate the in-depth evaluation of suspicious chemicals and run a public database in which consumers and professionals can find hazard information.

Not only new substances fall under the restrictions of REACH, but also known substances, even those that have been in use for a long time. The industry will face a huge task of testing and classifying all those substances. In a recent study by the European Chemical Bureau (ECB) it has been estimated that REACH will require 3.9 million additional test animals, if no alternative methods are accepted. Various toxicity types, in vivo mutagenicity and carcinogenicity are the endpoints that will require the largest number of test animals within REACH because no alternative in-vitro assays are available. The same study shows that it is possible to reduce the number of test animals significantly by utilizing existing experimental data in conjunction with (Q)SAR models. It is generally accepted that computer-aided classification will speed up the process, cut down on the costs and help avoiding excessive in-vivo testing.

The ECB has defined criteria for (Q)SAR models which where originally set forth by the Organisation for Economic Cooperation and Development (OECD): A QSAR model should be associated with

- a defined endpoint,
- an unambiguous algorithm,
- appropriate measures of goodness-of-fit, robustness and predictivity,
- a mechanistic interpretation, if possible.

Lazar clearly fulfills the first requirement, as every training database corresponds to a specific endpoint. Model building and prediction has been discussed in section 2 and is fully governed by an unambiguous algorithm.

In the multilinear model, a goodness-of-fit statistic is provided for each prediction in the form of a χ^2 value per degree of freedom (see Appendix B). Predictions are generally robust because of the distance weighting in instance-based methods, i.e. a local model is not easily affected by distant compounds. Small changes in the database, for example, will only influence a small number of predictions. Predictivity measures for two popular databases were given in section 3.1.

Furthermore, Lazar derives its predictions (rational conclusions) automatically from existing experimental data and presents them in an interpretable and traceable manner. A mechanistic interpretation can be obtained from the common substructures between the query structure and the neighbours: they are significant for the endpoint under inspection and can be used by human experts to identify toxicity mechanisms. The confidence index defines an applicability domain with high predictivity, which can, after thorough evaluation by a toxicological expert, be used as a replacement for in-vivo bioassays. The expert may conclude that the prediction is reliable enough to omit further bioassays if there is a high prediction confidence and neighbours that probably act by similar mechanisms.

For the future, it will be of great importance for toxicological experts to be able to compare different (Q)SAR models in an objective validation framework. This will give the researcher the possibility to reliably validate prediction results, as comparable results from different models generally increase stability and reliability in data mining approaches. To this end, important steps forward have to be taken in the predictive toxicology community: a common database format (adhering to databases with different access policies and legal status), systematic data quality (incorporating chemical structures) and the unambiguous identification of chemicals to name only a few important ones.

6 Conclusion

6.1 Summary

We have seen that instance-based learning methods are well suited for heterogenous training databases because they bypass the problem of fitting a model to a set of non-congeneric compounds (i.e. compounds that act by different chemical mechanisms). The extent to which this happens depends of course on the features selected and the distance metric applied. Structural properties naturally relate to chemical mechanisms and are therefore a good choice. They can also be mined from the data in a complete and statistically sound fashion with corresponding significance values and without the need to resort to literature.

Using (linear) fragments as structural information often yields an enormous amount of descriptors for a chemical structure. Lazar identifies the relevant features with statistical tests, which guarantees completeness. It also assigns a significance value to each fragment.

Lazar derives predictions using k-nearest-neighbour techniques. The distance metric for compounds (similarity) builds directly on the significance of linear fragments and is an obvious way to extend the structural approach. Since distance weighting is used, the number of neighbours can be quite large, limited solely by computational aspects such as runtime and memory. The neighbours can be inspected by human experts, who in turn can pin down physicochemical mechanisms that may trigger the endpoint under investigation.

Three models have been developed and evaluated using leave-one-out crossvalidation for two popular, publicly available databases. All models are using the same confidence index to describe the reliability of their predictions. Concerning accuracy, the multilinear model shows a consis-

tent improvement over the median model. Here, accuracy decreases nearly monotonically together with confidence, which is a clear indicator for the stability of the distance metric. The cluster model could not make up for the information loss due to compression with the incorporation of more fragments. The transformed data seems to lack linear relations for which possible reasons have been discussed.

It has been shown that Lazar fulfills the reqirements specific to instance-based learning approaches. Other current (Q)SAR models have been reported and Lazar ranks with these approaches. The basic requirements for industrial and/or regulatory use have been considered and it has been shown that Lazar meets those requirements.

6.2 Future Work

The next generation of Lazar is supposed to additionally support quantitative features, which will hopefully further improve accuracy and also provide a more usable basis for principle components analysis. The quantitative features should also be mined automatically and completely using data mining methods.

A fast and convenient graphical user interface is to be designed. There are two versions, suitable for different scenarios: a web interface accessible with any graphical browser for beginners and occasional users and a binary program for a local workstation for heavy use and confidential data.

A common validation framework for different (Q)SAR models will make it easy to compare results of different approaces quickly and easily. The author wants to participate in this process and Lazar will offer an interface to such a framework as soon as a definition exists.

A Installation

Use the following instructions to install Lazar using the multilinear model as prediction model.

These are build instructions for Ubuntu, a Debian-based Linux distribution, release 6.10 (*Edgy Eft*). For other Debian-based distributions the process should differ only slightly.

A.1 Preliminaries

To compile Lazar, install *subversion* and *g++* compiler along with helper programs.

```
box:~$ sudo aptitude install subversion build-essential
```

`lazar-tools` is a collection of programs that come with Lazar and perform statistical evaluations on Lazar LOO data. I assume you have `PERL` installed already. Now install the XML parser.

```
box:~$ sudo perl -MCPAN -e shell
cpan> install Bundle::CPAN
cpan> reload cpan
cpan> install XML::Parser
cpan> quit
```

We also need software from the *R Project for Statistical Computing*.

```
box:~$ sudo aptitude install r-base
```

A.2 Libraries

GNU Scientific Library (GSL). Lazar uses GSL routines and data structures internally for many purposes.

```
box:~$ sudo aptitude install libgsl0-dev
```

OpenBabel, the open source chemistry toolbox: Compile from source.

```
box:~$ wget http://mesh.dl.sourceforge.net/sourceforge/ \
        openbabel/openbabel-2.0.1.tar.gz
box:~$ tar zxvf openbabel-2.0.1.tar.gz && cd openbabel-2.0.1
```

Follow the directions given in `INSTALL`. Create some symbolic links:

```
box:~$ cd /usr/local/include/openbabel-2.0
box:/usr/local/include/openbabel-2.0$ sudo find \
                -name ''*.h'' -exec ln -s {} \;
```

A.3 Compiling Lazar

Check includefiles and libraries:
the GSL includefiles should reside in `/usr/include/gsl/`, the Openbabel includefiles in `/usr/local/include`. The shared library `libopenbabel.so` should be found in `/usr/local/lib` and `libgslcblas.so` and `libgsl.so` in `/usr/lib/`. If all looks good, check out the Lazar source code from the subversion repository and compile:

```
box:~$ svn checkout svn://www.in-silico.de/lazar/\
                branches/quantitative
box:~/quantitative$ cd quantitative/ && make
```

If you experience problems, check the `Makefile`: Check the path to the OpenBabel includefiles in the `CXXFLAGS` variable. The default is:

```
CXXFLAGS = -g -O2 -I/usr/local/include/openbabel-2.0/ -w
```

After successful compilation, you should have the executable `lazar` in the current directory.

A.4 Running Lazar

To run Lazar using the multilinear model follow the steps outlined below.

Change to the directory where you want LOO to be run. Example: you want to run Lazar on the FDAMDD dataset. First extract all linear fragments from the database, then run Lazar.

```
box:~/quantitative$ cd data/fdamdd060410/data
box:~/quantitative/data/fdamdd060410/data$ make
box:~/quantitative/data/fdamdd060410/data$ cd ../validation
box:~/quantitative/data/fdamdd060410/validation$ ./run_lazar
```

Before running `make` or `run_lazar`, check the `Makefile` in the `data` directory, specifically the `TOOLDIR` variable, for the correct path to the `lazar-tools` directory. Check `run_lazar` in the `validation` directory, specifically the the `LAZDIR` variable, for the correct Lazar root directory (default: `/home/<user>/quantitative`).

The run can take several hours. When finished, Lazar will have created some files, the most important being `.loo` (contains output produced directly by the Lazar binary) and `.summary` (contains error statistics). Accuracy and prediction plots will also have been created. See Appendix B.

Adjust the file `ad.pl` in the current directory to try different applicability domains (set the variable `ad_threshold`).

B File Formats

B.1 Input

Lazar uses several input files, which together form the training database:

- `.smi`: contains a row for each compound in the database in SMILES format. Each compound is assigned a unique number. Example:

```
1691    C1=CC(C=O)=CC(OC)=C1OCCCCCC
1692    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
...
```

- `.linfrag`: contains a row for each fragment occuring in the database in SMARTS format. Each fragment is assigned a list with the compound numbers it occurs in. This file corresponds to step 1. in the main Lazar workflow (see figure 3). As this information never changes for a given database, it can be extracted beforehand for efficiency reasons. Example:

```
C       [ 0 1 3 4 5 6 7 ... ]
c       [ 0 1 3 4 5 6 7 ... ]
...
```

- `.act`: contains a row for each compound in the database, identified by it's number. Each compound is assigned it's quantitative activity value, together with a text string identifying the endpoint of the experimental source. Example:

```
1691    LC50_mg 2.67
1692    LC50_mg 73.3
1694    LC50_mg 1.92
...
```

B.2 Output

Lazar outputs several files in XML format:

- `.loo`: When LOO is performed, this file receives all the data for every prediction made. For every prediction, after identifying the compound by line number, id and SMILES code, it lists all neighbours; after that follow the prediction results, i.e. database activity, confidence, prediction and chi^2 value of the fit[1]. Example:

```
<compound>
        <endpoint>LC50_mg</endpoint>
        <line_nr>1</line_nr>
        <id>1691</id>
        <smiles>C1=CC(C=O)=CC(OC)=C1OCCCCCC</smiles>
        <neighbour>
                <similarity>0.59375</similarity>
                <line_nr>262</line_nr>
                <id>1968</id>
                <smiles>O=CC1=CC(OCC)=C(O)C=C1</smiles>
                <activity>1.9425</activity>
        </neighbour>
        [ ... more neighbours ...]
        <chi_sq>0.02587</chi_sq>
        <prediction>1.38351</prediction>
        <confidence>0.0443589</confidence>
        <db_activity>0.426511</db_activity>
</compound>
```

[1] Some additional information has been omitted for a better overview

- `.summary`: Contains the performance indices for LOO. It gives the number of predictions, root mean-squared error ($rmse$) and R^2 coefficient for all predictions and for the predictions in the applicability domain[m]. Example: After that follows - in descending order of con-

```
<summary>
<all>
        <nr_err>574</nr_err>
        <rmse>0.94</rmse>
        <r_sq>0.44</r_sq>
</all>
<within_ad>
        <nr_err>67</nr_err>
        <rmse>0.76</rmse>
        <r_sq>0.71</r_sq>
</within_ad>
```

fidence - the errors for the predictions. This is used to calculate the regression sum of squares. Example:

```
<cumulative_accuracies>
        <compound>
                <id>621</id>
                <smiles>O=C(C(C1=CC=C(C=C1)Cl)[...]</smiles>
                <prediction>-3.72125</prediction>
                <confidence>0.763393</confidence>
                <db_activity>-2.29243</db_activity>
                <error>1.42882</error>
        </compound>
        [ ... more predictions ... ]
</cumulative_accuracies>
```

C Detailed Validation Results

The following provides detailed validation results, broken down to models and applicability domains. Shown are plots of confidence vs. cumulative R^2 and of predicted activity vs. database activity.

[m] Some additional information has been omitted for a better overview

C.1 EPAFHM

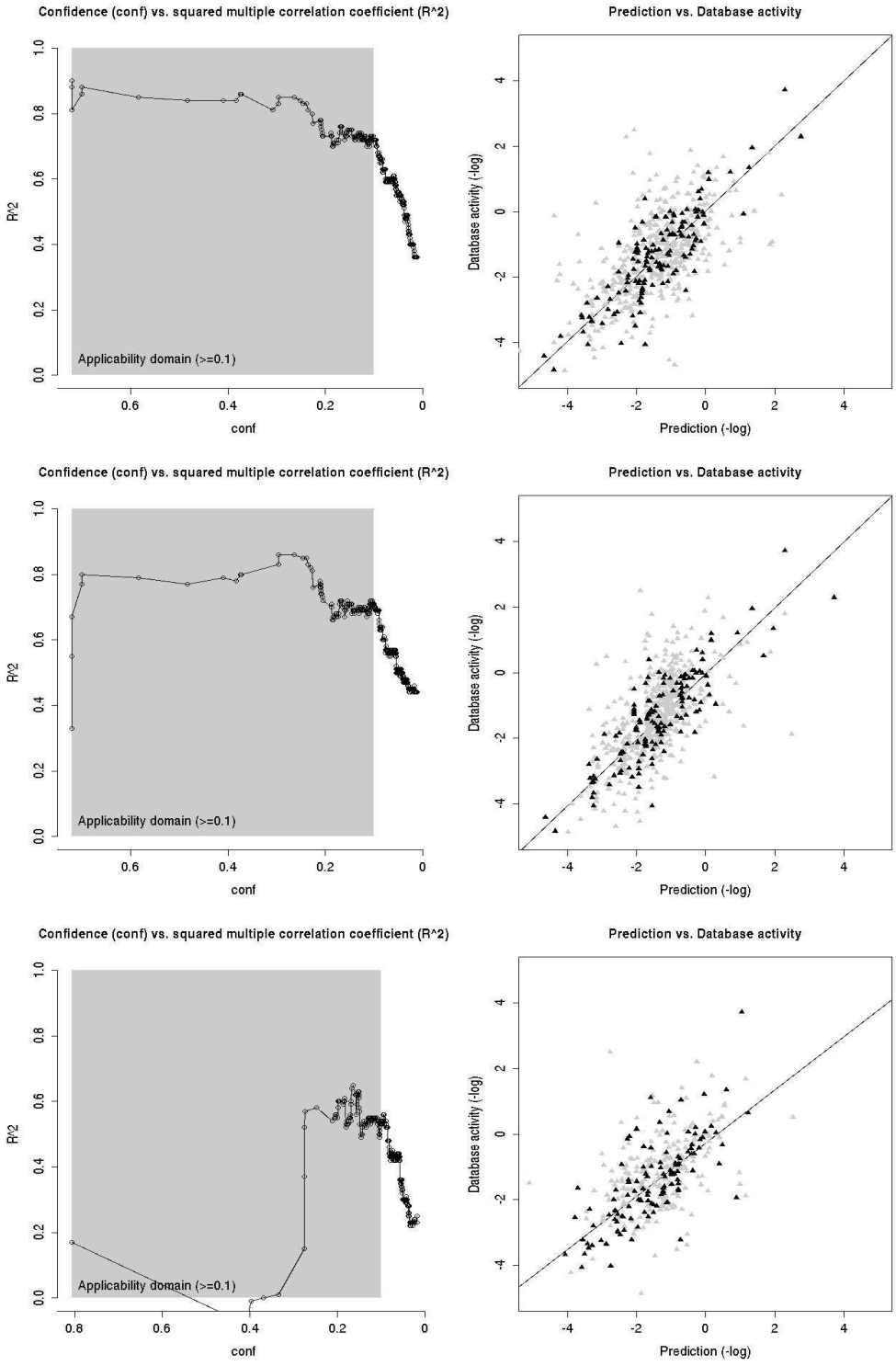


Figure 15: Multilinear model (top), median model (middle) and cluster model (below) for EPAFHM with applicability domain 0.1

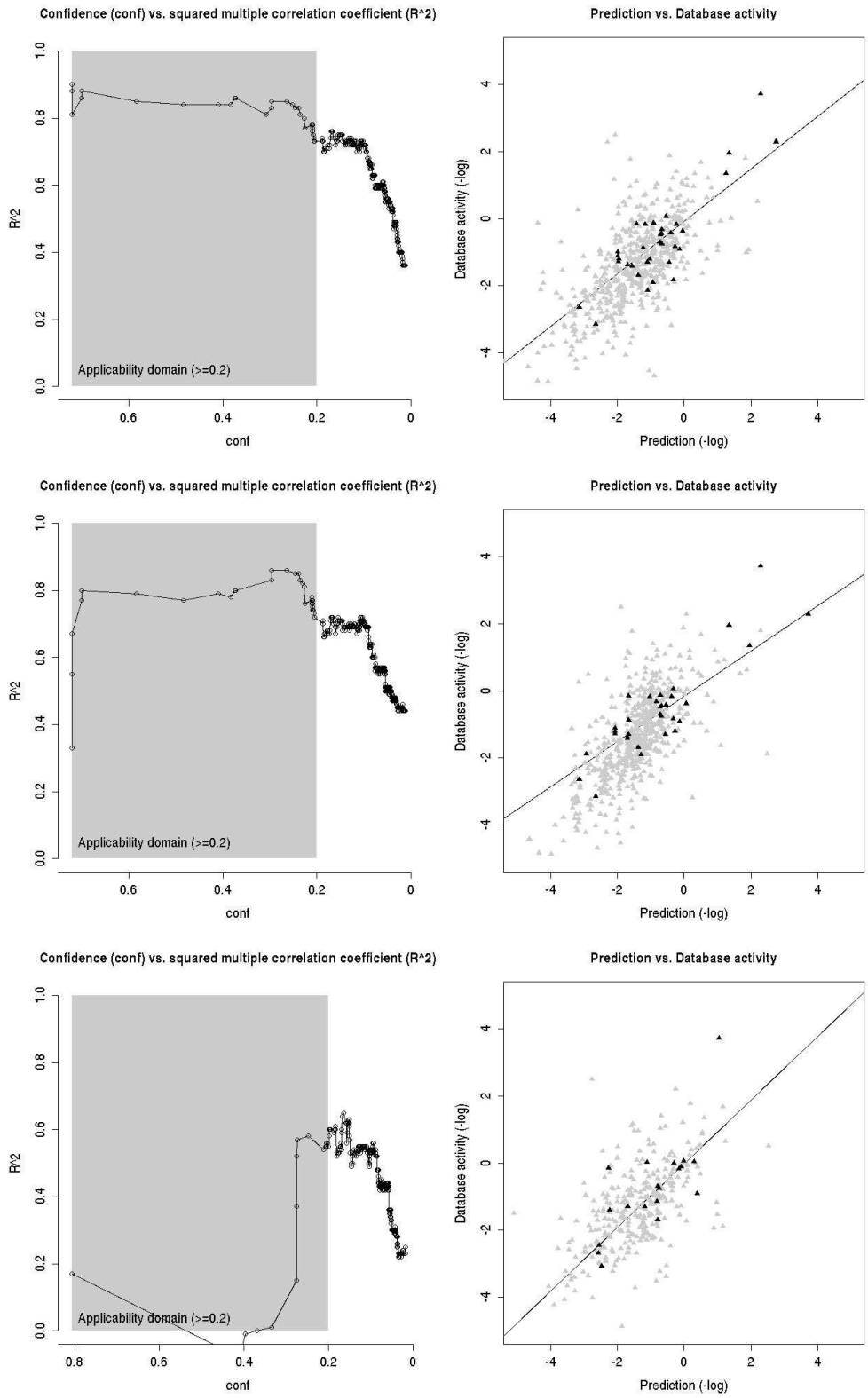


Figure 16: Multilinear model (top), median model (middle) and cluster model (below) for EPAFHM with applicability domain 0.2

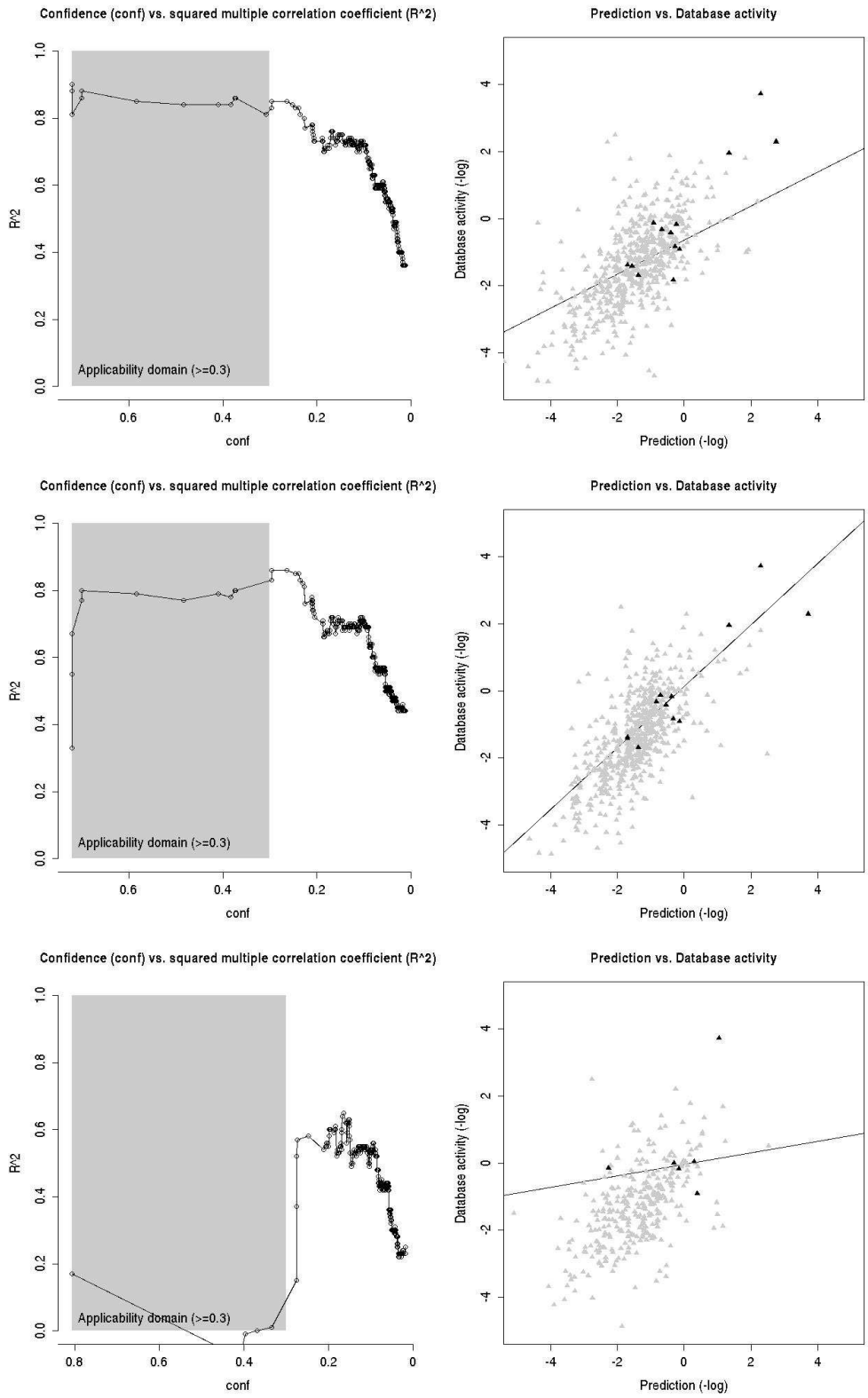


Figure 17: Multilinear model (top), median model (middle) and cluster model (below) for EPAFHM with applicability domain 0.3

C.2 FDAMDD

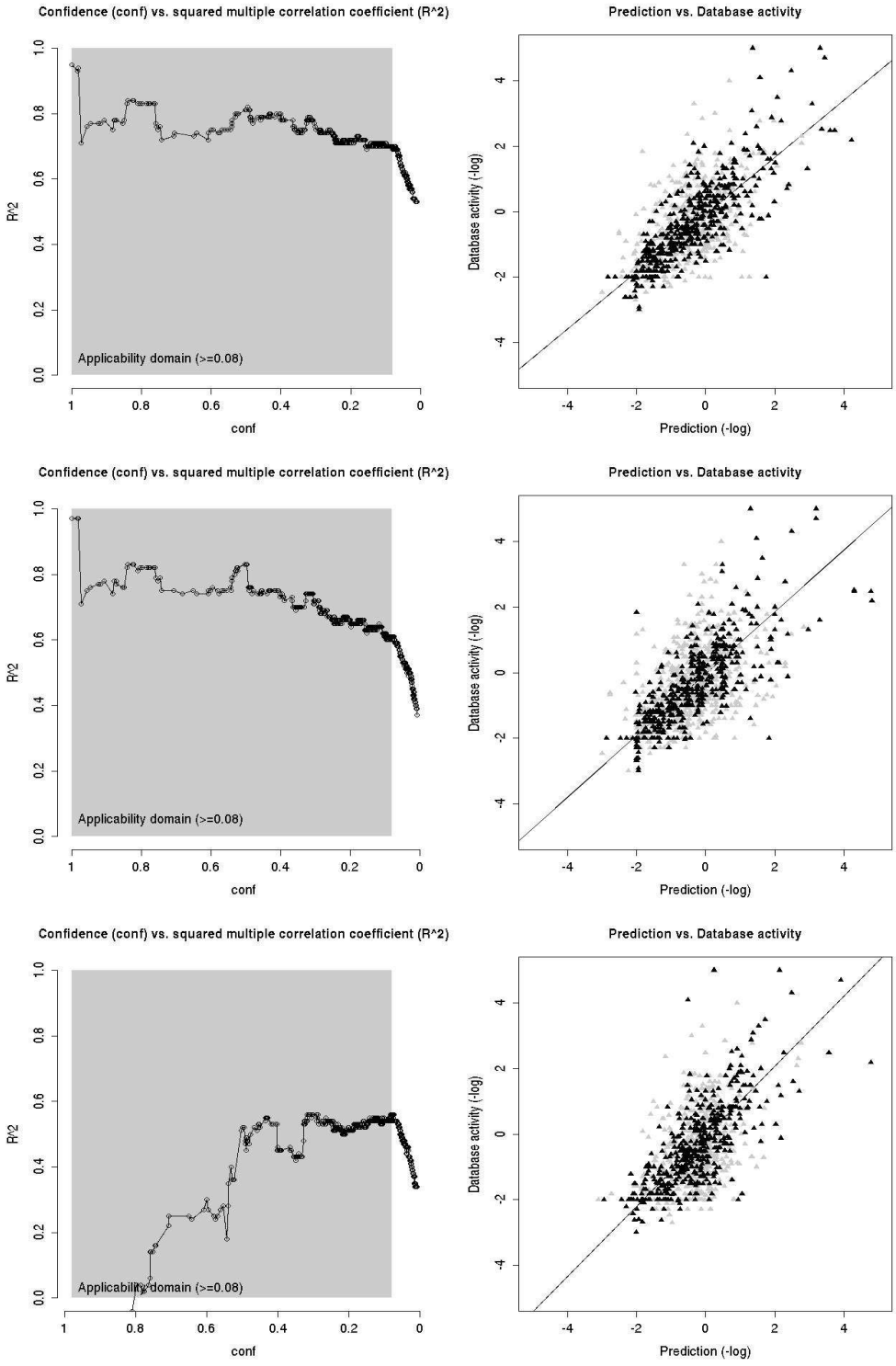


Figure 18: Multilinear model (top), median model (middle) and cluster model (below) for FDAMDD with applicability domain 0.08

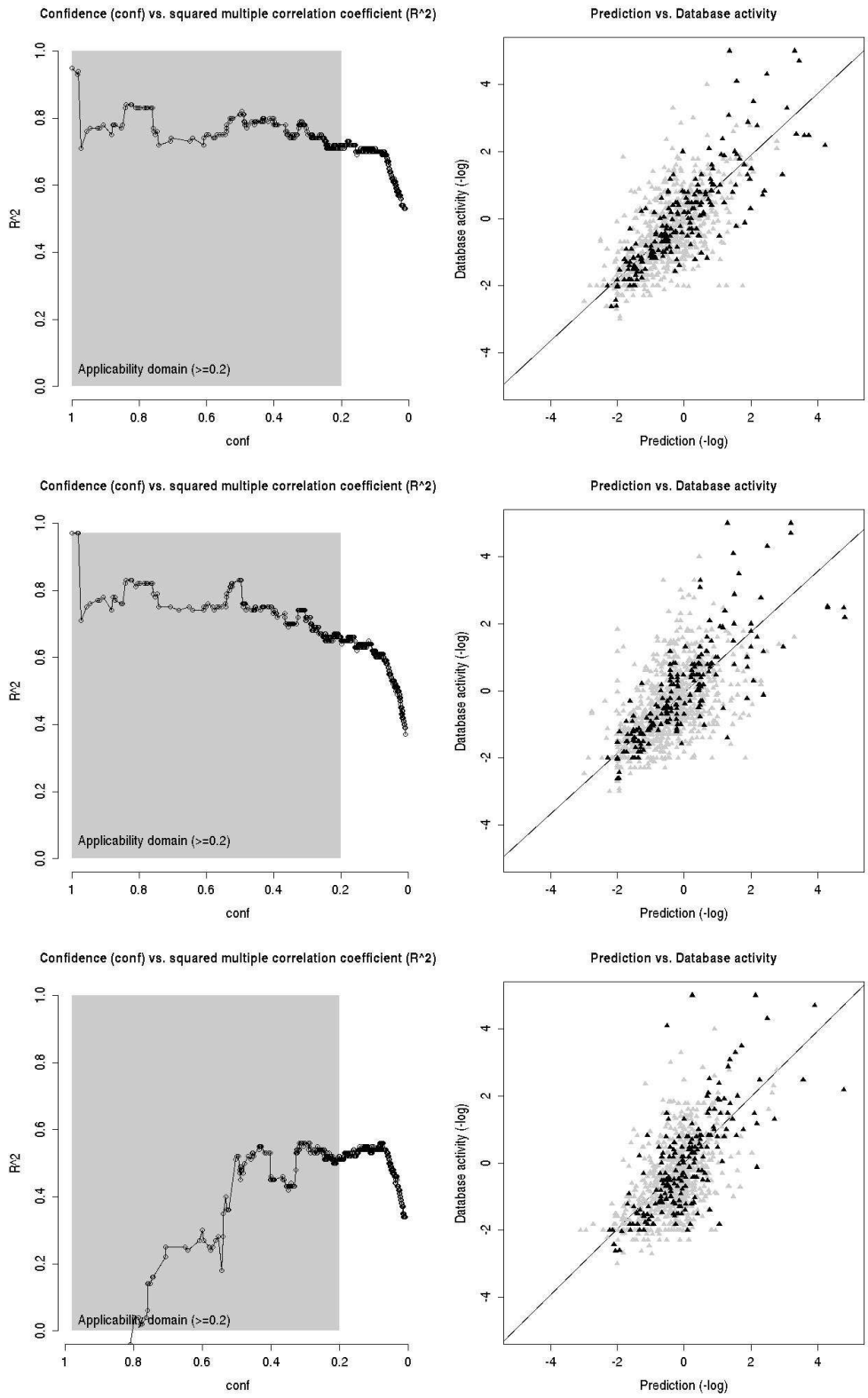


Figure 19: Multilinear model (top), median model (middle) and cluster model (below) for FDAMDD with applicability domain 0.2

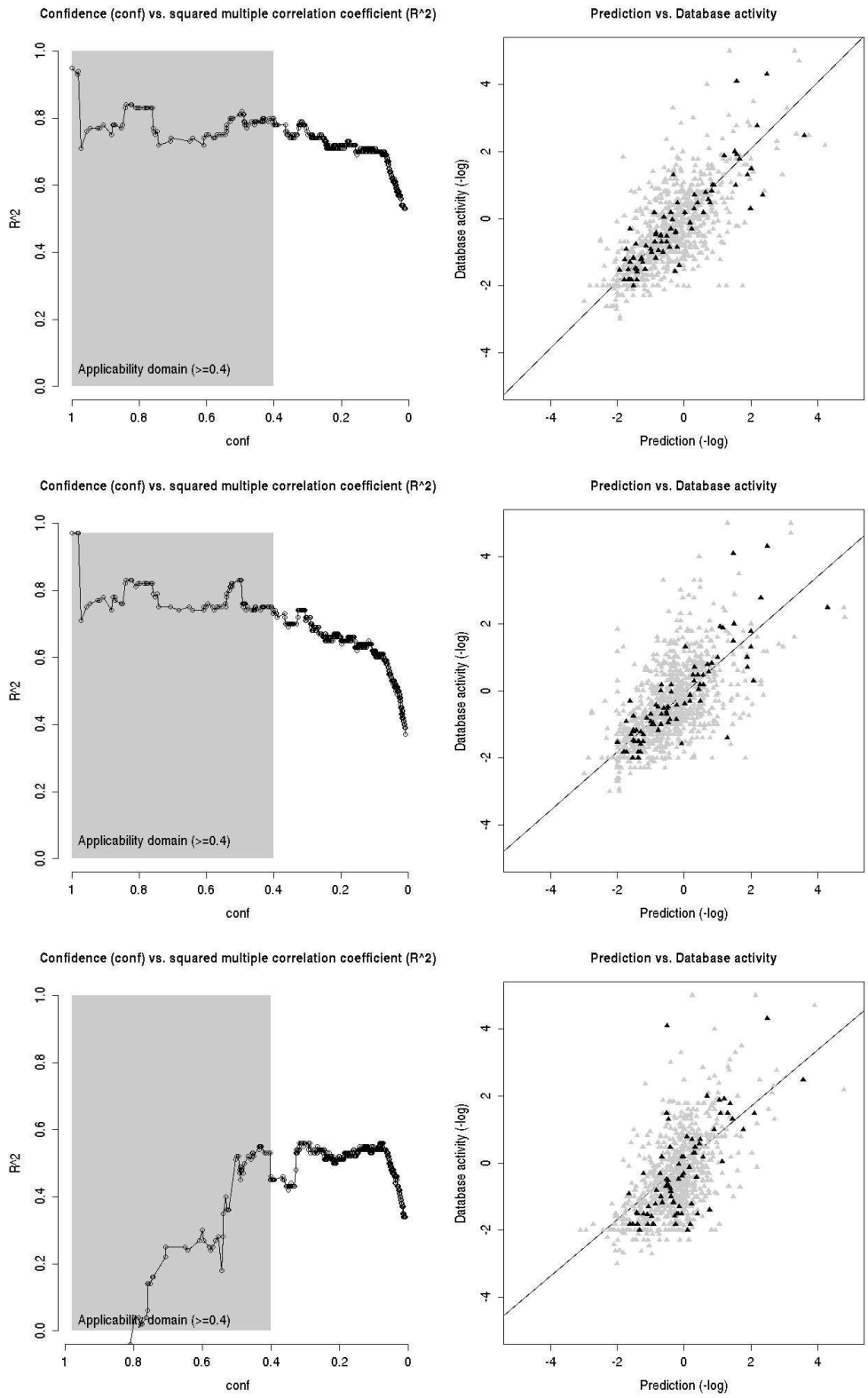


Figure 20: Multilinear model (top), median model (middle) and cluster model (below) for FDAMDD with applicability domain 0.4

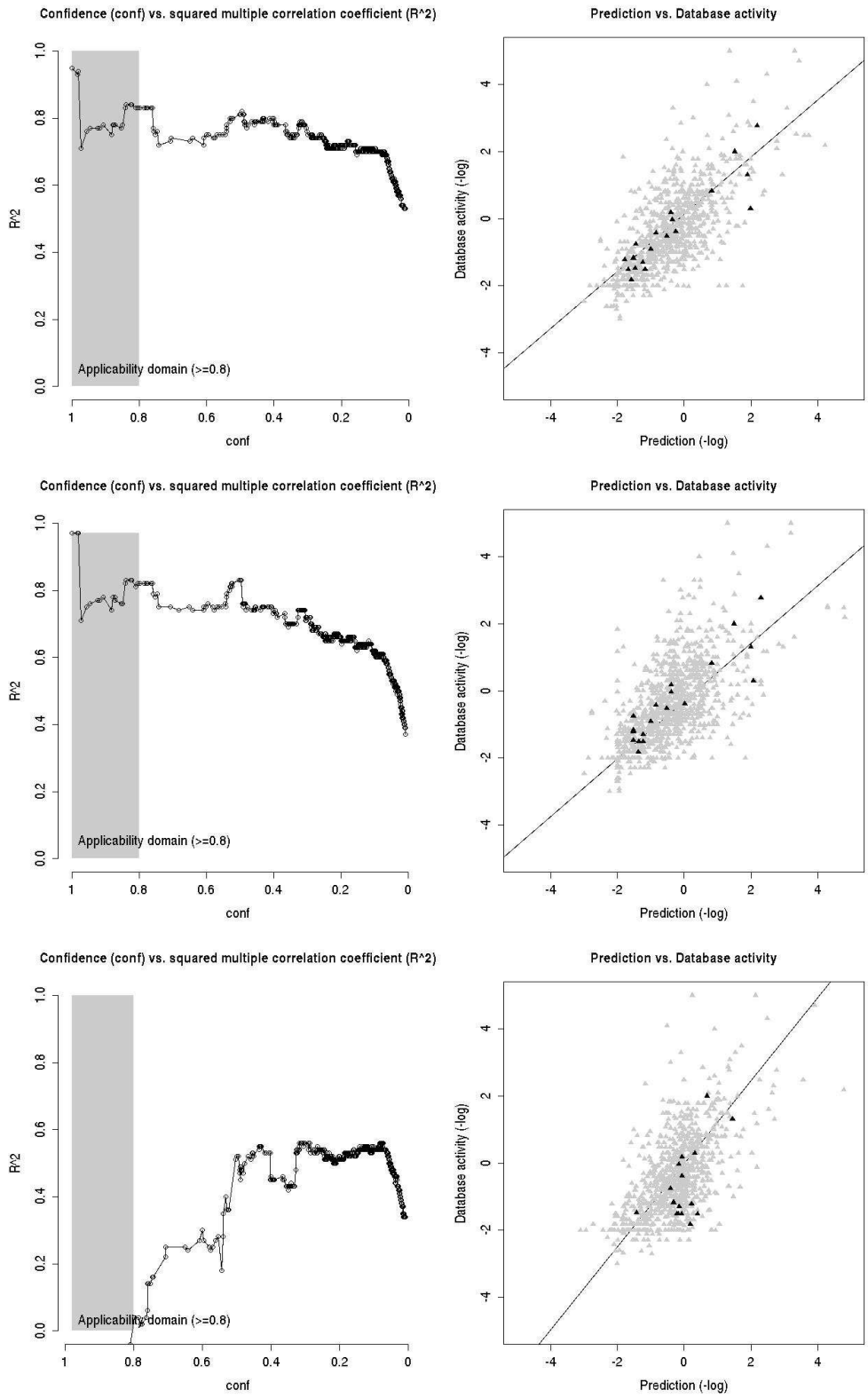
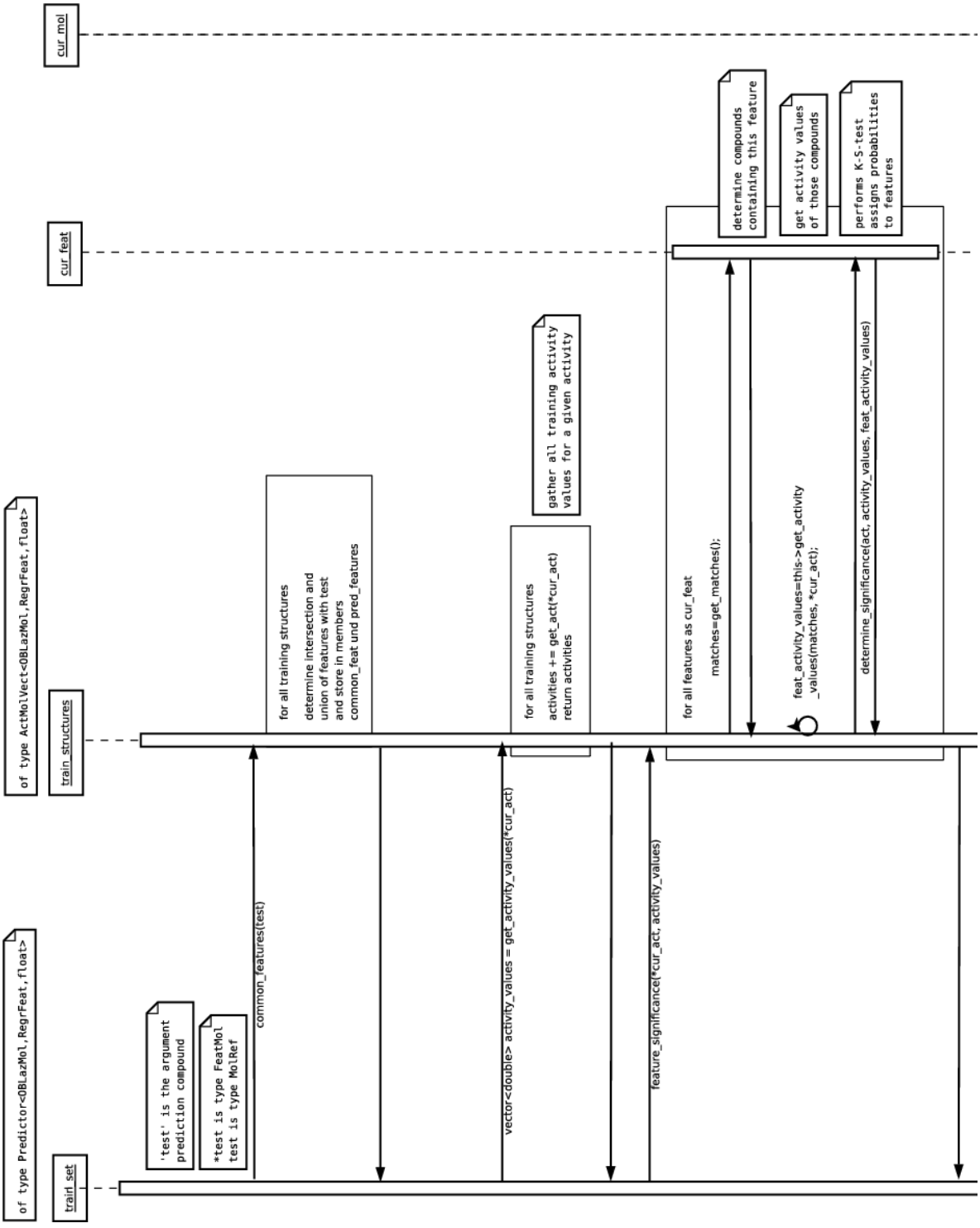
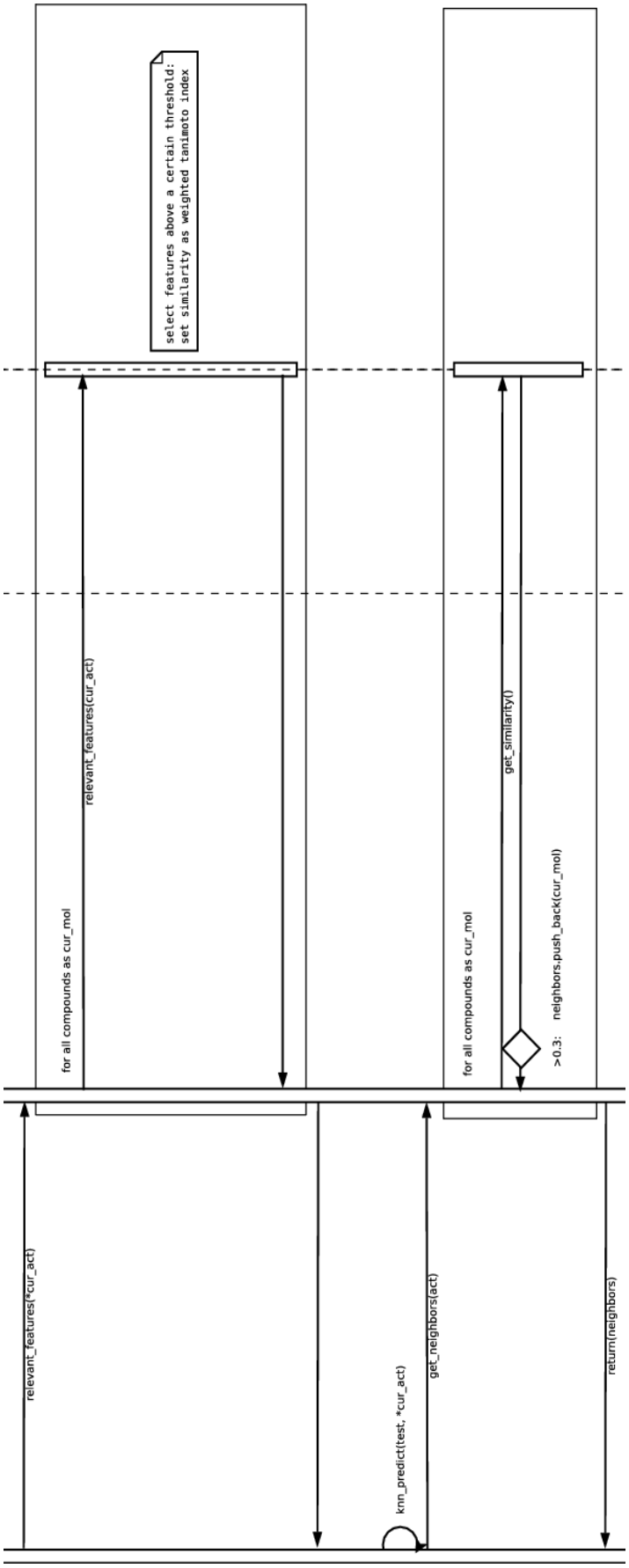


Figure 21: Multilinear model (top), median model (middle) and cluster model (below) for FDAMDD with applicability domain 0.8

D Lazar Prediction Process

The following UML diagram shows the Lazar algorithm, detailing the calculation of significant features (first page) and neighbours (second page).


of type Predictor<OBLazMol,RegrFeat,float>
of type ActMolVect<OBLazMol,RegrFeat,float>
train set
train structures
cur feat
cur mol
'test' is the argument prediction compound
*test is type FeatMol test is type MolRef
common_features(test)
for all training structures determine intersection and union of features with test and store in members common_feat und pred_features
vector<double> activity_values = get_activity_values(*cur_act)
for all training structures activities += get_act(*cur_act) return activities
gather all training activity values for a given activity
feature_significance(*cur_act, activity_values)
for all features as cur_feat
matches=get_matches();
determine compounds containing this feature
feat_activity_values=this->get_activity _values(matches, *cur_act);
get activity values of those compounds
determine_significance(act, activity_values, feat_activity_values)
performs K-S-test assigns probabilities to features


select features above a certain threshold:
set similarity as weighted tanimoto index
relevant_features(cur_act)
for all compounds as cur_mol
get_similarity()
for all compounds as cur_mol
>0.3: neighbors.push_back(cur_mol)
relevant_features(*cur_act)
knn_predict(test, *cur_act)
get_neighbors(act)
return(neighbors)

References

[CL04] Mark T.D. Cronin and David J. Livingstone. *Predicting Chemical Toxicity and Fate*, chapter 8, pages 151–170. CRC Press, 2004.

[EPA] U.S. Environmental Protection Agency EPA. DSSTOX database. http://www.epa.gov/ncct/dsstox/index.html.

[EWJK99] Donald V. Eldred, Cara L. Weikel, Peter C. Jurs, and Klaus L.E. Kaiser. Prediction of fathead minnow acute toxicity of organic compounds from molecular structure. *Chem. Res. Toxicol.*, (12):670–678, December 1999.

[Hel04] Christoph Helma. *Predictive Toxicology*, chapter 1. Taylor and Francis, 2004.

[Hel06] Christoph Helma. Lazy structure-activity relationships (lazar) for the prediction of rodent carcinogenicity and salmonella mutagenicity. *Molecular Diversity*, (10):147–158, 2006.

[InC] Inchi, the international chemical identifier. Available online at http://www.inchi.info.

[Jol02] Ian T. Jolliffe. *Principal Components Analysis*. Springer, 2 edition, 2002.

[JWD00] C.A. James, D. Weininger, and J. Delany. *Daylight theory manual - Daylight 4.71*. Daylight Chemical Information Systems, 2000. http://www.daylight.com.

[KDH01] Stefan Kramer, Luc DeRaedt, and Christoph Helma. Molecular feature mining in hiv data. 2001.

[KH04] Stephan Kramer and Christoph Helma. *Predictive Toxicology*, chapter 7, pages 223–254. Taylor and Francis, 2004.

[Mit97] Tom M. Mitchell. *Machine Learning*, chapter 8. WCB/McGraw-Hill, 1997.

[MY01] Todd M. Martin and Douglas M. Young. Prediction of the acute toxicity (96-h lc50) of organic compounds to the fathead minnow (pimephales promelas) using a group contribution method. *Chem. Res. Toxicol.*, (14):1378–1385, March 2001.

[NAHL04] S.P. Niculescu, A. Atkinson, G. Hammond, and M. Lewis. Using fragment chemistry data mining and probabilistic neural networks in screening chemicals for acute toxicity to the fathead minnow. *SAR and QSAR in Environmental Research*, 2004.

[Ö04] Thomas Öberg. A qsar for baseline toxicity: Validation, domain of application, and prediction. *Chem. Res. Toxicol.*, (17):1630–1637, June 2004.

[PNW06] M. Pavan, T.I. Netzeva, and A.P. Worth. Validation of a qsar model for acute toxicity. *SAR and QSAR in Environmental Research*, (2):147–171, April 2006.

[PTV93] William H. Press, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C*, chapter 14.3. Cambridge University Press, 1993.

[PVG05] Ester Papa, Fulvio Villa, and Paola Gramatica. Statistically validated qsars, based on theoretical descriptors, for modeling aquatic toxicity of organic chemicals in pimephales promelas (fathead minnow). *J. Chem. Inf. Model.*, (45):1256–1266, May 2005.

[REA06] Regulation (ec) no 1907/2006 of the european parliament and of the council of 18 december 2006 concerning the registration,

evaluation, authorisation and restriction of chemicals (reach). *Official Journal of the European Union*, 2006.

[Smi02] Lindsay I Smith. A tutorial on principal components analyis. February 2002.

[TLL95] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *J. Chem. Inf. Comput. Sci.*, 1995.

Index