

How Wrong Can We Get? A Review of Machine Learning Approaches and Error Bars

Anton Schwaighofer^{*1}, Timon Schroeter², Sebastian Mika³, Gilles Blanchard¹

^{*} *anton@first.fraunhofer.de*

¹ Fraunhofer FIRST, Kekuléstraße 7, 12489 Berlin, Germany

² Technische Universität Berlin, Department of Computer Science, Franklinstraße 28/29, 10587 Berlin, Germany

³ idalab GmbH, Sophienstraße 24, 10178 Berlin, Germany

Abstract

A large number of different machine learning methods can potentially be used for ligand-based virtual screening. In our contribution, we focus on three specific nonlinear methods, namely support vector regression, Gaussian process models, and decision trees. For each of these methods, we provide a short and intuitive introduction. In particular, we will also discuss how confidence estimates (error bars) can be obtained from these methods. We continue with important aspects for model building and evaluation, such as methodologies for model selection, evaluation, performance criteria, and how the quality of error bar estimates can be verified. Besides an introduction to the respective methods, we will also point to available implementations, and discuss important issues for the practical application.

Keywords: machine learning, error bars, model building, parameter estimation, decision tree, support vector machine, Gaussian process

1 Introduction

Numerous approaches have been developed to increase the cost-effectiveness of the drug discovery process. One of them is the use of virtual screening techniques. Here, a large body of molecules is filtered and ranked by computational means, in order to identify those that have a high chance of having desirable properties. Traditionally, only the activity in the biological system (*i.e.*, receptor binding) has been considered. During recent years, most companies have started to consider ADME and other properties early in the development process.

Virtual screening can be based on crystal structures of target receptors (structure based virtual screening). If the crystal structure of the target protein is unknown, but the structure of a related protein with similar sequence and probably similar structure is available, this information can be used to estimate the structure of the target (homology modeling, [1]). Another alternative is to identify new active compounds by an analysis of known molecules with the desired activity. In this paper, we focus exclusively on this last, so-called ligand based, approach to virtual screening.

Starting with information about the biological activity of a number of compounds, how can we make accurate predictions for the biological activity of new compounds? In an abstract form, this question has been a major focus of research in machine learning for many years. In this paper, we will review some machine learning methods that have proven successful for predicting various properties of chemical compounds, such as biological activity, physico-chemical properties such as water solubility, or ADME properties. We will give theoretical background on some methods, and try to give some advice on the choice of methods and aspects of the practical application.

Clearly, machine learning methods of any complexity have their limitations. They all are based on establishing some form of statistical correlation between a description of the compound and its activity. This statistical model acts as a proxy for the complex biological or physico-chemical processes that lead to the observed activity. The prediction by the machine learning method can be quite accurate when predicting the activity of a compound where a lot of training data for similar compounds is available--and almost arbitrarily wrong when dissimilar structures are queried or the descriptors do not cover the complexity of the property to predict. For example, a model for water solubility will fail to predict the (very low) solubility of halogenated biphenyls, if none of its training structures contained this motif.

Thus, a key requirement for any form of machine learning model is to provide a confidence estimate for each individual prediction [2,3]. However, machine learning methods differ widely in how they can provide this confidence estimate. We will discuss these differences in an extra section.

Apart from these more methodological aspects, we will devote a section to important practical aspects such as parameter selection, model building and validation. We give pointers to a number of freely available implementations of the methods we discuss. The paper ends with a brief summary of some application examples.

2 Methods

There is a large number of different statistical and machine learning methods that is suitable for predicting properties of compounds (see *e.g.*, [4]). These range from simple (but often sufficient) linear methods to complex nonlinear predictors. Clearly, we cannot review all of these here, and instead focus on three methods in particular: Support vector regression (SVR, [5]), Gaussian process regression (GPR, [6]), and decision trees (DT, [7]). SVR and, in particular, GPR are methods that are very much in the focus of current machine learning research, and show close relationships. Trees, on the other hand, have been chosen as an example of a well established, yet very powerful and robust prediction technique.

In the subsequent sections, we will use \mathbf{x} to denote a vector of d descriptors that describe a particular compound. As the “training” data that we use to build a predictive model, we have N descriptor vectors, $\mathbf{x}_1, \dots, \mathbf{x}_N$, along with the experimental outcomes for the N compounds y_1, \dots, y_N . When necessary, we will use \mathbf{x}_* to denote the descriptor vector for a test compound, *i.e.*, a compound the model is being evaluated on. In general, we denote by $f(\mathbf{x})$ the output of a model on a compound represented by its descriptor vector \mathbf{x} .

2.1 Support Vector Methods

Support vector machines (SVM, [5,8]) for regression and classification are based on the principle of structural risk minimization. The principle is to first consider a class of functions, where each of them takes the descriptors as input, and outputs the property to predict. From the class of functions, we want to find the function that minimizes the error on the training data, where errors are measured by the so-called loss function. Clearly, a trade-off has to be made concerning the complexity of the functions we use: Using a complex function, it is possible to perfectly fit the training data, but most likely this function will not be able to generalize and predict properties of new, unseen compounds (over-fitting). On the other hand, when using a function that is too simple, we may not be able to fit the data reasonably well, again resulting in inaccurate predictions. Practically, choosing a function class with the right complexity can be achieved by regularization: For model fitting, a well established approach is to use the sum of the empirical loss on the training data and a penalty term for the complexity as objective function for learning.

In the subsequent section, we focus on support vector regression (SVR [5,9,10]) only. We will first describe the idea behind linear SVR and then generalize to the non-linear case by using kernel functions.

Given a vector \mathbf{x} of descriptors for a compound, the quantity of interest y will be predicted as $y = f(\mathbf{x})$. Linear SVR finds a predictor that, similar to ordinary linear regression, predicts $f(\mathbf{x}) = b + \mathbf{w}^T \mathbf{x}$, with a weight vector \mathbf{w} of length d and a bias term b . By \mathbf{w}^T we denote the transpose of vector \mathbf{w} . Weights \mathbf{w} are chosen such that the sum of empirical error on the training data and norm of the weight vector $\|\mathbf{w}\|^2$ are minimal.

A typical feature of SVM is the concept of sparseness: The solution is built up from only the most relevant training data (the support vectors). To achieve this, Vapnik [5] devised the so-called ϵ -insensitive loss function. Here, prediction errors smaller than ϵ will not be taken into

account, whereas errors larger than ϵ contribute linearly to the overall error. In contrast, in linear regression or GP regression (see Sec. 2.2), all errors contribute quadratically to the overall error. Using the ϵ -insensitive loss function is thus considered to be more robust against outliers in the training data. Fig. (1) illustrates the principle of the ϵ -insensitive loss.

INSERT Figure (1) about here

With the training data for N compounds (see above), linear SVR can be formulated as the following optimization problem (see Ch. 9 of [8])

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i \quad (1)$$

subject to $|f(\mathbf{x}_i) - y_i| \leq \epsilon + \xi_i$ and $\xi_i \geq 0, i = 1, \dots, N$

The objective function to be minimized is the weighted sum of regularization term $\|\mathbf{w}\|^2$ and the overall sum of errors made. The proportion of the error that exceeds ϵ is captured by the variables ξ_i , which in turn contribute linearly to the objective function.

Nonlinear SVR

An extension to non-linear regression models is possible by employing a transformation that is sometimes called the “kernel trick” (the transformation actually dates back to the 1960s, see Sec. 2.2.2 of [8]). The above objective function can be written in a form where the vector of descriptors for each compound, \mathbf{x}_i , enters only via scalar (dot) products, such as $\mathbf{x}_i^T \mathbf{x}_j$. These scalar products can then be replaced by a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, that *implicitly* maps the descriptors into a high-dimensional feature space and computes the scalar product there. Evaluating the kernel function with two arguments, each of them being a vector in descriptor (input) space, we obtain as a result the scalar product of the projections in the high-dimensional feature space.

With this change, prediction of the non-linear SVR model on a test compound \mathbf{x}_* takes on the form of a weighted sum of kernel functions,

$$f(\mathbf{x}_*) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) k(\mathbf{x}_i, \mathbf{x}_*) + b \quad (2)$$

Coefficients α_i and α_i^* are obtained by solving the following optimization problem (see Ch. 9 of [8] for the derivation):

$$\max_{\alpha, \alpha^*} \frac{1}{2} (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*)^T K (\boldsymbol{\alpha} - \boldsymbol{\alpha}^*) - \epsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) + \sum_{i=1}^N y_i (\alpha_i - \alpha_i^*) \quad (3)$$

subject to $\sum_{i=1}^N (\alpha_i - \alpha_i^*) = 0$ and $\alpha_i \geq 0, \alpha_i^* \geq 0, \alpha_i \leq C, \alpha_i^* \leq C$

In this form, K denotes the matrix of pairwise kernel evaluations $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. $\boldsymbol{\alpha}$ and $\boldsymbol{\alpha}^*$ are the vectors of coefficients $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)$ and $\boldsymbol{\alpha}^* = (\alpha_1^*, \dots, \alpha_N^*)$.

The optimization problem in Eq. (3) and Eq. (3) can now be used in off-the-shelf routines for solving constrained optimization problems. A specific property of support vector methods in general is that they produce sparse solutions, where a fraction of coefficients in Eq. (2) is zero, that is, $(\alpha_i - \alpha_i^*) = 0$. Thus, despite originally expressing the solution Eq. (2) in terms of all training data, the SVR prediction for large training data can be computed quite efficiently. Note that the degree of sparseness depends on the complexity of the modeling task, the chosen model parameters C , ϵ , and the parameters of the kernel function. Simple modeling tasks tend to give high sparsity (many coefficients are zero), similarly high ϵ and small C usually give high sparsity.

For large scale problems, the matrix K is too big to fit into memory, thus specific properties of the optimization problem (in particular the sparseness) have to be used to obtain an efficient training procedure [12,13,14].

Regularization

In SVM classification models, the choice for minimizing the norm $\|w\|^2$ can be argued for by maximizing the margin that separates the two classes and as such gives rise to low complexity solutions. In linear SVR models, minimizing $\|w\|^2$ can be interpreted as choosing the flattest function that can explain the data reasonably well. When extending SVR to nonlinear predictions via the kernel function, $\|w\|^2$ relates to the degree of smoothness of the function f . Each kernel function corresponds to a regularization operator P (see [9,10], Ch. 9 of [8]), that, for example, describes the derivatives and thus the smoothness of the function f . With this operator, $\|w\|^2$ can be re-written as an inner product $\langle Pf, Pf \rangle$, where Pf is the regularization operator applied to the prediction function f . Minimizing $\|w\|^2$ thus chooses the smoothest function that can explain the data well, where smoothness is implicitly defined by the kernel resp. its equivalent regularization operator.

A kernel function that is often used with SVM/SVR methods is the so-called RBF kernel,

$$k(\mathbf{x}, \mathbf{x}') = \exp(-u\|\mathbf{x} - \mathbf{x}'\|^2) \quad (4)$$

where $\|\cdot\|^2$ denotes Euclidean norm. The parameter u in this covariance function needs to be chosen beforehand, see Sec. 5.1. However, there exists a plethora of other standard and problem specific kernel functions (a non-representative selection is [15,16,17,18,19,20,21]).

As a last point, we would like to remark that SVR can as well be formulated with different ways of measuring the error, that is, loss functions other than the ϵ -insensitive loss we have introduced above [9,10]. When using these loss functions, the sparseness of the solution is typically lost, thus the sum in Eq. (2) will have non-zero contributions for all training data. Such variants of SVR will be much more computationally demanding than the sparse standard SVR, both during training and prediction.

2.2 Gaussian Process Regression

Gaussian process (GP) models have their origin [6] in the field of Bayesian statistics. The key idea is to initially consider a probability distribution over functions, and use the principles of Bayesian inference to obtain the *a posteriori* probability distribution over functions after having observed data [22,23].

INSERT Figure
(2) about here

The principles behind GP models are illustrated in Fig. (2). Before observing experimental data, we assume that the relationship between the descriptor and the property of interest is modeled by an infinitely large family of functions (25 functions plotted in Fig. (2)(a)). The family of functions is described by a Gaussian stochastic process [23]. As we have not seen any data yet, we can consider all functions as roughly equally likely to be the right predictor for the property of interest. Fig. (2)(b) illustrates the situation after having observed experimental data (the data for 4 compounds is marked by crosses). We additionally account for measurement uncertainty, and draw a small “tunnel” around the experimental value. To combine this with our prior assumptions, we can now discard (or put lower weight on) all functions that do not pass through the tunnels around the observed data. To predict the property for a new molecule, we can simply average over the functions remaining in the pool (this gives the dark grey line in Fig. (2)(c)) and read off the value corresponding to the new molecule's descriptors. In this framework, we can readily obtain an estimate for the prediction error by calculating the standard deviation of the functions remaining. The 2σ environment for all descriptor values on the x -axis is marked in light grey in Fig. (2)(c). Note that uncertainty is small close to the observed data but not zero, since measurements are assumed to be noisy. The uncertainty increases far from the observed data, thus providing an implicit measure for the domain of applicability of the model.

Of course, explicitly enumerating the infinitely large prior pool of functions is not feasible. In the following, we describe how the inference process outlined above can be computed efficiently.

Let f be a function that depends on a vector \mathbf{x} of d molecular descriptors and outputs the endpoint of interest, $f(\mathbf{x})$. The key assumption here is that the functional values $f(\mathbf{x}_1), \dots, f(\mathbf{x}_N)$ for any finite set of N points form a joint N -variate Gaussian distribution (hence the name Gaussian process). The distribution is characterized by its mean (which we assume to be zero) and the $N \times N$ covariance matrix, the elements of which are given by the covariance function,

$$\text{cov}(f(\mathbf{x}), f(\mathbf{x}')) = k(\mathbf{x}, \mathbf{x}'), \quad (5)$$

which has a role similar to the kernel functions in SVMs (see Sec. 2.1) and other kernel based learning methods. Assuming zero mean makes sense if we shift all experimental value before training to have zero mean, and shift back accordingly when making predictions. Next, we assume that all experimental data are noisy, and relate the measured values y_1, \dots, y_N to their “true” property, $f(\mathbf{x}_i)$, via

$$y_i = f(\mathbf{x}_i) + \varepsilon \quad (6)$$

where ε is measurement Gaussian noise with standard deviation σ .

With these model assumptions, applying Bayes' rule (see Ch. 2 of [23] or [22]) already leads to the prediction $f(\mathbf{x}_*)$ for a new compound \mathbf{x}_* : The predicted property follows a Gaussian distribution with mean $\bar{f}(\mathbf{x}_*)$ and standard deviation $\text{std } f(\mathbf{x}_*)$, with

$$\bar{f}(\mathbf{x}_*) = \sum_{i=1}^N \alpha_i k(\mathbf{x}_i, \mathbf{x}_*) \quad (7)$$

$$\text{std } f(\mathbf{x}_*) = \sqrt{k(\mathbf{x}_*, \mathbf{x}_*) - \sum_{i=1}^N \sum_{j=1}^N k(\mathbf{x}_i, \mathbf{x}_*) k(\mathbf{x}_j, \mathbf{x}_*) H_{ij}} \quad (8)$$

Coefficients $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)$ are found by solving a system of linear equations, $(K + \sigma^2 I)\boldsymbol{\alpha} = \mathbf{y}$, with $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, I as the unit matrix, and $\mathbf{y} = (y_1, \dots, y_N)$. For the standard deviation, H_{ij} are the elements of the matrix $H = (K + \sigma^2 I)^{-1}$. Similarly to SVMs, the solution for the coefficients $\boldsymbol{\alpha}$ is unique.

Clearly, the GP modeling approach hinges on the choice of a suitable covariance function, Eq. (5). The class of valid covariance functions for GP models is identical to the class of valid kernel functions for SVM methods (the class of positive definite functions, [8,23]). Thus, all general and problem specific kernel functions listed in Sec. 2.1 can also be used as covariance functions for GP modeling. A standard choice for GP models on vectorial descriptor data is the RBF covariance function, Eq. (4). Note, however, that the choice of covariance function not only influences the accuracy, but also the quality of the error bars (see Sec. 5.4). In all of our applications [22,24], we found that best results can be achieved with the rational quadratic covariance function [23],

$$k(\mathbf{x}_1, \mathbf{x}_2) = s \left(1 + \sum_{i=1}^d u_i (\mathbf{x}_1^{(i)} - \mathbf{x}_2^{(i)})^2 \right)^{-\nu} \quad (9)$$

Here, we denote by $\mathbf{x}^{(i)}$ the i .th descriptor in \mathbf{x} . GP models with the rational quadratic covariance function are usually as accurate as those with RBF covariance, Eq. (4), and additionally give well fitting error bars. Parameters s and u_i can be learned from the data using marginal likelihood, as will be described in Sec. 5.1.2.

Large Scale GPR

When using Gaussian process models as outlined here, memory demand scales with the second power of the number of training compounds. In typical software environments, this limits the use of standard Gaussian process models to a maximum of around 5000 points. As for SVR models, large scale learning requires a few modifications. Whereas for SVR the sparsity of the solution allowed large scale learning, GPR needs to make modifications of the implied prior [25]. Alternatively, a local modeling approach can be used, where training data is clustered (for example, using k-means clustering) and local models can be fit to each cluster. When applying the model, predictions for each cluster GP model are generated, and the prediction with the highest confidence (smallest error bar) is chosen.

Individual Measurement Noise

As a further point, it is worth mentioning that the GP model allows to either share the same measurement uncertainty σ (see Eq. (6)) for all measurements, or set it differently for, *e.g.*, groups of compounds. In our work on solubility prediction [22], we used a noise variance σ_1 for those compounds where only one experimental value was available, σ_2 for compounds with two experiments, and σ_3 for compounds with three or more experimental values. The

exact values for σ_1 , σ_2 , and σ_3 need not be known in advance, but can be learned from the data using marginal likelihood (see Sec. 5.1.2).

Uniqueness of SVR and GPR Solutions

GPR and SVR share one appealing property, namely that with a given covariance (kernel) function, the solution is unique and can be found efficiently. In the case of GPR, learning (model fitting) amounts to solving a system of linear equations to obtain the weights α in Eq. (7). The optimization problem for SVR, Eq. (3), is a so-called convex problem, thus the solution can be found efficiently as well. In both cases, the solutions are unique due to the positive definiteness of the covariance (kernel) function. This contrasts with other non-linear prediction techniques (in particular neural networks) which are prone to local minima.

2.3 Decision Trees

2.3.1 Definition of a Decision Tree

Decision trees [7] are a method for classification and regression, where the function that computes the decision has the form of a binary tree following the principle of the “20 questions” guessing game. At the root of the tree is the first question we ask about the object (the compound we wish to predict for). Depending on the answer (“yes” or “no”), we follow the left or right branch leaving the root and reach a new node of the tree, where we find the second question to ask, and so forth, until we reach a leaf of the tree; each distinct leaf contains a specific output decision (that is, a prediction for the property of interest y) which is then returned as the result.

The nature of the questions to ask can vary depending on the studied problem. A typical question when the input data \mathbf{x} is a numerical vector is of the form “is the k .th descriptor $\mathbf{x}^{(k)}$ bigger than a threshold value t ”? An example decision tree is shown in Fig. (3).

One reason for the popularity of single decision trees is their ease of interpretation and representation: it is easy from the structure of the tree to see not only the decision itself, but also *why* this precise decision was reached. Because the decision is taken after a succession of yes/no logical tests, decision trees are cousins of symbolic, rule-based learning methods. For some classical references on the subject, see [7,26,27].

INSERT Figure (3) about here

2.3.2 Building a Single Decision Tree

In order to learn an appropriate decision tree structure based on a set of labeled training data (descriptor vectors $\mathbf{x}_1, \dots, \mathbf{x}_N$ with according experimental values y_1, \dots, y_N), a general building principle is as follows: if we look at the set of training objects \mathbf{x}_s that reached a particular leaf s of the tree, the set of labels y_s for this subset of objects should be as homogeneous or “pure” as possible, so that we would be reasonably sure that a new object reaching this node will have a y value similar to the training objects present in that node. This way we can be reasonably confident about the decision that we take. The impurity of the labels in a subset S of training data can be measured in several ways. In the case of regression, a natural criterion to consider is the variance of the subset,

$$\text{var}(y_S) = \frac{1}{|S|} \sum_{i \in S} (y_i - \bar{y}_S)^2$$

where $\bar{y}_S = \frac{1}{|S|} \sum_{i \in S} y_i$ is the average y value in the subset S . When trees are used for classification, other classical impurity indices are considered such as the entropy and the Gini index [26]. In all cases, the criterion is positive and values closer to zero indicate a better purity.

Based on the impurity criterion, the tree is built sequentially: at each internal node, one chooses the question splitting the data into two sub-nodes such that the reduction in average impurity after the split is as large as possible (wherein each sub-node is weighted by the number of data points it contains). This way, a tree can be built by recursively applying this greedy impurity reduction principle.

The second important concern is when to stop growing the tree and declare that a node will not be split further and therefore become a decision leaf. Obviously, there is a potential overfitting problem here: without caution, we could split the data so many times that there is only one remaining training example per decision leaf. Such a leaf would appear perfectly pure, but of course decisions taken based on this single example are highly unreliable. Therefore, one must reach a compromise between purity of the leaf nodes and number of examples they contain. The simplest rule is to stop and not split a node further when there are less than n training examples left, say $n = 5$. Another, slightly more complicated, but generally preferred, method, is to first grow a large tree T_0 by splitting the data as many times as we can, and then prune this tree back by finding a subtree that realizes some tradeoff between average weighted purity and complexity, as measured by the tree size $|T|$ (the number of leaves),

$$\min_{T \text{ subtree of } T_0} \sum_{s: \text{leaves of } T} |s| \text{var}(y_s) + \lambda |T| = \min_{T \text{ subtree of } T_0} \sum_i (y_i - T(\mathbf{x}_i))^2 + \lambda |T| \quad (10)$$

where $T(\mathbf{x})$ denotes the y value predicted by the tree for observation $T(\mathbf{x})$ (that is, the average value for the training examples that reached the same leaf as \mathbf{x}). The justification for the above equation is that for a fixed tree structure T , the number of leaves $|T|$ is exactly the number of free parameters, and the above tree selection equation is then identical to a model selection using penalized least squares, a well-known principle in statistics. From this point of view, Mallows' C_p heuristic [28] suggests taking $\lambda = 2\sigma^2$, where σ^2 is the noise variance. If σ^2 can be estimated or is known from *a priori* information, this gives a good default value, which can possibly be refined using cross-validation on a local grid.

2.3.3 Multiple Decision Trees

Single decision trees have the advantage of being easy to use and to interpret. One significant disadvantage is that, in terms of raw prediction performance, they are generally outperformed by other, more elaborate techniques, and are quite unstable (minor changes in the training set can have a strong influence on the learned tree). To alleviate this problem, the most popular procedure is to build *multiple trees* T_1, \dots, T_m , a “forest”. The final output is then obtained by (possibly weighted) averaging of the outputs of the individual trees,

$$f(\mathbf{x}_*) = \sum_{i=1}^m g_i T_i(\mathbf{x}_*)$$

with weights g_i that satisfy $\sum_{i=1}^m g_i = 1$ (in the classification setting, the above is replaced by weighted voting). This is a particular case of so-called ensemble methods for machine learning. There exist three main methods for constructing multiple trees:

- Bagging [29], where the tree building procedure is applied repeatedly to different bootstrap samples (draw at random with replacement) from the original training set.
- Boosting [30] is a generic ensemble learning method that alternates learning and re-weighting of the objects in the training data. The examples for which the prediction is poor are given a higher weight for the next round and *vice versa*.
- Random Forests [31,32] combine bagging with a modification of the tree learning algorithm: each time a node s is split into two sub-nodes, instead of choosing among all possible questions for splitting the data, a random subset Q_S is first drawn from the pool of possible questions, and then the best question within Q_S in terms of impurity reduction criterion is selected. The size of Q_S can be tuned by the user.

Bagging was one of the first ensemble learning methods. Boosting or Random Forest are now preferred as they have been reported to be generally more accurate. In particular, it has been reported that the performance of Random Forest is consistently quite good for high-dimensional data across a variety of practical situations. It often has comparable performance to more elaborate methods, while being generally easier to use, which makes it an often recommended off-the-shelf learning method.

As a final note, we remark that the generally substantially higher performance of multiple tree methods comes at the price of losing the direct interpretability of a single tree. However, besides good performance, multiple trees are still a very useful tool to analyze the importance of descriptors, as it is possible to compute statistical reports about how often questions asking for a certain descriptor are used across the ensemble of trees [32]. As an additional advantage, the bootstrap re-sampling method used both in Bagging and Random Forest allows to compute so-called “out-of-bag estimates” (close in spirit to cross-validation, but computed on the fly) for various quantities of interest such as the overall error rate.

3 Estimating the Domain of Applicability of Models

A typical challenge for statistical models in the chemical space is to adequately determine the domain of applicability (DOA), *i.e.*, the part of the chemical space where the model's predictions are trustworthy. To this end several classical approaches exist.

- *Range based methods* are based on checking whether descriptors of test set compounds exceed the range of the respective descriptor covered in training [33,34]. A warning message is raised when this occurs. The domain of applicability is thus the smallest axis-parallel rectangle that contains all training data. An obvious

improvement is achieved by *geometric methods* that estimate the convex hull of the training data [2]. Here, the domain of applicability is smallest polyhedron that contains all training data. Note that both these methods are unable to detect “holes” in the training data, that is, regions that are only scarcely populated with data. Similarly, single outlier values in a descriptor can mislead these approaches.

- If experimental data for some new compounds are available, error estimates based on the *library approach* [35] can be used. By considering the closest neighbors in the library of new compounds and the prediction errors for those compounds, it is possible to get a rough estimate of the bias for the respective test compound.
- *Probability density distribution based methods* are another way of estimating the model reliability [2], by checking whether the test set compound is in a region of high density of training data. However, it should be noted that high dimensional density estimation is in itself a difficult task, in particular since the behavior of densities in high dimensions may be completely counterintuitive [36].
- *Distance based methods* (including the extrapolation measures) [37,2,3,38,33] consider one of a number of distance measures (Euclidean distance, Mahalanobis distance, etc.) to calculate the distance in descriptor space between the test compound and its closest neighbor(s) in the training data. Another way of using distance measures is to define a threshold and count the number of training compounds closer than the threshold. Hotellings test or the leverage [33] rely on the assumption that the data follows a Gaussian distribution in descriptor space and compute the Mahalanobis distance to the whole training set. As pointed out by Tetko [3], the relevance of individual descriptors depends on the property to be predicted, hence property specific distances should be used instead.

All these methods estimate a form of closeness to the training data. They are a form of post-processing that is decoupled from the model building process, and can be used with any kind of model.

Instead of asking “how close is a test compound to the training data?”, one might as well ask directly “how accurate is the prediction for a test compound?”. Such forms of estimating the domain of applicability need to be integrated into the model building process.

- One well known strategy is to consider *ensemble methods*, where a number of models is trained on different sets of data. Typically the sets are generated by sampling from a larger set of available training data. The models in the ensemble will tend to agree in regions of the descriptor space where a lot of training compounds are available and will disagree in sparsely populated regions. Alternatively, the training sets for the individual models may be generated by adding noise to the descriptors, such that each model operates on a slightly modified version of the whole set of descriptors. In this case the models will agree in regions where the predictions are not very sensitive to small changes in the descriptors and they will disagree in descriptor space regions where the sensitivity with respect to small descriptor changes is large. (However, it must be noted that the chemical feasibility of such an approach strongly depends on the nature of the descriptor and its connection to the property to predict. There may be cases where a small change in the descriptor indeed reflects a change in the molecule that would completely change its property). Ensemble methods can be based on various forms of classifiers, ensembles of neural networks [3,38,39,40,37] and decision trees (see Sec. 2.3) [34,37] are most commonly used. Associative neural networks [3] are a hybrid approach, where an ensemble of neural networks is used to

establish a form of DOA measure that is based on correlation in the space of ensemble predictions.

- Estimates for the prediction accuracy are an integral part of the idea behind *Bayesian models* [41,42], such as the Gaussian process models introduced in Sec. 2.2. Generally, the idea of Bayesian statistics is to treat all quantities involved in modeling (such as model parameters, but also experimental values) as random variables. By applying the rules of Bayesian inference, the *a priori* assumptions about parameters are combined with the experimental data, to obtain the *a posteriori* knowledge. Hence, such models naturally output a probability distribution, instead of the point prediction in conventional learning methods. Regions of high predictive variance not only indicate that a compound is outside the domain of applicability, but can also indicate regions of contradictory or scarce measurements.

In some cases, individual members of an ensemble can also produce an estimate of the reliability of their own prediction. In the case of trees, the (lack of) accuracy of a single tree's prediction can be estimated by the variance of the training examples that are present in the corresponding decision leaf. These individual variance estimates can then be averaged over the ensemble to predict the variance of the ensemble output. In our experiments with trees, the overall predictive variance was the average of the ensemble variance (as described above) and the mean variance in each decision leaf. We found that this heuristic generally gave very satisfactory results.

3.1 One Dimensional Examples

Fig. (4) gives a graphical intuition as to how three different methods of error estimation (distance based, ensemble based, and Bayesian) work. We consider a one-dimensional case, where we learn the sine function (shown as a dashed grey line in each subplot). The available training data (marked by crosses) are randomly chosen points, on which we evaluate the sine function and add Gaussian measurement noise to obtain the y values.

INSERT Figure (4) about here

The SVR model, Fig. (4)(a), produces a good fit to the data and also extrapolates well. We use SVR with a distance based error estimate, by computing the Mahalanobis distance to the closest training point and use a linear function to map from distance to error bar. In this example, this gives a slightly conservative (large) error bar in the region close the training points, but somewhat underestimates the errors in the extrapolation region.

The random forest, Fig. (4)(b), does not produce a very appealing (yet reasonable) fit to the data, since the prediction function is by construction piecewise constant and thus not smooth. We use an ensemble based error estimate, which, in this example, gives an acceptable error bar estimate in the vicinity of the training points. Yet, when predictions far from the training points are sought, it underestimates the errors. Note that, in this one-dimensional example, the random forests method is equivalent to bagging.

The Gaussian process model, Fig. (4)(c) also gives a good fit to the data, and extrapolates reasonably. From the GP model, we readily obtain the Bayesian error estimate, the predictive standard deviation, Eq. (8). In this example, the predictive standard deviation is small in the region close to the training points, but increases strong enough in the extrapolation region.

Extrapolation Behavior

A crucial point about all DOA measures is their behavior when making predictions that are far from the training data. Clearly, for such predictions, the DOA measure or error bar should indicate that the model is most likely completely wrong. When looking at the SVM with its distance based error estimate in Fig. (4)(a), we see that the error bar increases linearly with increasing distance, since we have used a linear function that maps from distance to error bar (an exponential function can also be used here). The error bar would thus approach infinity for compounds that are infinitely far away from the training data.

For the random forest, error bars (spread of ensemble predictions or a related quantity, see Sec. 2.3) in regions far from the training data depend on the ensemble predictions on the boundaries of the training data. In the toy example in Fig. (4)(b), the error bars in regions left and right of the training data have approximately the same size, yet this is not the general rule. Note furthermore that, for ensemble based error estimation to be stable, enough members must be present in the ensemble. Plotting the toy example in Fig. (4)(b) with an ensemble of only two trees, for example, leads to error bars that are very tight in the extrapolation region (unless bagging by chance chooses a very favorable split).

In the case of GP models, error bars (the predictive standard deviation in Eq. (8)) for compounds far from the training data approach a value given by the amplitude of the kernel function, namely $\sqrt{k(\mathbf{x}_*, \mathbf{x}_*)}$. This, in turn, is (roughly) equal to the standard deviation of the experimental values y_1, \dots, y_N . The prediction of a GP model for a completely unknown substance (very or infinitely far from the training data) is thus the mean of experimental values in the training data, with the standard deviation of experimental values as the error bar. This can be considered as the most conservative guess amongst those that do take the experimental values into account. When asked for a prediction on very distant points, a plain distance based error bar estimate returns “the value will be between $-\infty$ and $+\infty$ ”, whereas the GP model is slightly more optimistic and returns “the value will be in the range spanned by the training data”.

As a last point here, note that there are some machine learning methods with model-based error estimates where the predicted standard deviation approaches zero for compounds that are far from the training data. For example, the so-called finite linear models [25] make the wrong *a priori* assumption that the function to predict only has variability in the region of training data. This contrasts with, *e.g.*, Gaussian processes, which assume that the function to predict has variability all over the descriptor space.

4 Descriptors

Machine learning methods can be used to predict properties of chemical compounds in a number of ways. It is, for example, possible to construct kernels for SVR or GPR that operate directly on the structural formula (molecular graph, [17]). Despite the theoretical appeal of directly comparing two molecular graphs, it limits the choice of possible prediction methods to the class of kernel-based methods. In this article we will instead focus on descriptor based modeling. In this case, a number of descriptors is calculated from the structural formula or a 3D representation of the molecule. Most machine learning and statistical methods can then directly work on this representation, where each molecule is described by a vector (of fixed length) that contains all the descriptors.

Another advantage of this strategy is that prior knowledge can easily be incorporated. If we know, for example, that aqueous solubility is closely related to $\log P$, and already have a well performing predictive model for $\log P$ available [24,43], the calculated $\log P$ can simply be included in each molecule's vector of descriptors [22,44].

Descriptor Selection

More than 3000 different molecular descriptors have been developed [45]. The less is known about a specific modeling problem, the less likely it is that we will be able to decide *a priori* which descriptors are suitable for the task at hand. Therefore, it is common to start with a large set of descriptors and reduce this set using different descriptor selection algorithms [33,46] (in the machine learning literature they are called feature selection methods).

Descriptor selection techniques are particularly important for methods that run into trouble when used with large numbers of correlated descriptors. These methods clearly benefit from a reduction of the number of descriptors, and can also profit from an orthogonalizing pre-processing such as principal component analysis. Some authors conclude that descriptor selection is an essential part of the modeling process (see [47] and references therein).

In our own work, we found that descriptor selection must not necessarily be carried out. Support vector machines and Gaussian processes can handle large numbers of descriptors, even if a lot of them are correlated. Random forests are, as well, robust with respect to the presence of correlated descriptors due to the random selection of questions Q_S , see Sec. 2.3. Excellent results have been achieved with SVM models with more than 100,000 descriptors [48]. We found that it is often possible to reduce the number of descriptors dramatically without reducing the prediction performance [22,43,44]. Liu *et al.* [48] report that descriptor selection often has a negative impact on the classification performance of SVMs. Other authors [49] report performance gains by using sophisticated descriptor selection techniques. Yet, due to the high variability of these results it remains unclear to us whether these performance gains are statistically significant.

On the other hand, descriptor selection can bring serious drawbacks when also considering error estimates. By projecting all compounds to a lower-dimensional descriptor space, more and more compounds look similar, when they actually are not similar. Thus, it is getting more difficult to detect cases where the domain of applicability is left. For example, when using Gaussian process models with a smaller and smaller number of descriptors, we found that the quality of the predicted error bars decreases faster than the actual prediction performance [24,44,43]. The models still produce accurate predictions most of the time, but they fail to identify molecules for which predictions are not reliable.

A similar argument holds for range-based and distance based methods to estimate the domain of applicability [2,37,44,43]. Imagine building a model from a training set of compounds that does not contain any sulfur atoms, using descriptors that include counts of functional groups. In this case, all counts of S-containing groups will be zero. Any sensible descriptor selection method will eliminate these counts. When this model is now applied to compounds that do include sulfur, it cannot warn the user by assigning lower confidence to the predictions for compounds with the unknown functional groups.

In conclusion, reducing the number of descriptors by descriptor selection algorithms is not necessary for all learning methods. This holds in particular for methods where the number of descriptors is not directly related to the number of free parameters, such as SVR or GP models

(the kernel function Eq. (4) has one free parameter, irrespective of whether there are 10 or 10^6 descriptors). Retaining more descriptors than necessary for achieving good prediction performance can have further advantages, like allowing more accurate error bars (for Gaussian process models) or other estimates of the domain of applicability.

Descriptor Pre-Processing

Depending on their nature, descriptors can span different numerical ranges (for example, 0 to 1 for descriptors indicating the existence of a fragment, or $-\infty$ to $+\infty$ for eigenvalue descriptors). Methods such as SVR and GPR compute distances in descriptor space, which will then clearly be dominated by descriptors with large numerical range, irrespective of their significance for the property to predict. An often necessary pre-processing step is thus to normalize descriptor values, such that all descriptors have a mean value of 0 and a standard deviation of 1. For fragment counts and other descriptors that are natural numbers, an additional non-linear transformation $x \leftarrow \log(1 + x)$ before normalization often has beneficial effects: The structural difference between compounds that contain 0 resp. 1 occurrences of a fragment is larger than that between compounds that contain 50 resp. 51 occurrences of the fragment. The log transformation follows this intuition.

5 Model Building and Evaluation

All of the methods presented in Sec. 2 depend on different parameters that we have assumed to be known. Thus, a suitable strategy for selecting these parameter needs to be implemented.

It should be emphasized that the step of parameter (model) selection is crucial. We believe it is fair to say that a simple modeling approach, combined with careful parameter selection, can easily outperform a sophisticated model with sloppy parameter selection. Similarly, parameter selection should be performed carefully to avoid over-fitting. Tuning parameters until the overall performance (*e.g.*, on a specific test or validation set) is optimal is a very tempting way of achieving satisfactory results. It may, however, lead to extremely biased results that seem more promising than they in fact are. In the next sections, we will detail how model selection should be performed to (i) obtain a model with high performance while (ii) maintaining a realistic estimate for the generalization error.

5.1 Parameter Selection

5.1.1 Nested Cross-Validation

The methods we briefly described in the previous sections differ fundamentally in terms of how the model parameter can be chosen. The linear SVR model, Eq. (1), has two parameters that need to be known before learning, ε and the weight C of the regularization term. When using non-linear SVR with kernels, parameters of the kernel function need to be known beforehand as well, such as the width parameter u in Eq. (4). Decision Trees have a number of parameters that influence the tree building process, such as the maximum tree depth, the number of trees when using multiple trees, or the pruning parameter λ in Eq. (10).

For both methods, all parameters need to be selected by a search procedure that has the cross-validation (CV) performance as its objective function. The process of model building is typically combined with a process of evaluation, which leads to a nested cross-validation scheme, as illustrated in Fig. (5). Search can clearly be performed in different ways, in its simplest form we can use grid search. For the grid search, a set of possible values needs to be defined for each parameter, leading to a large set of possible parameter combinations for the model. Out of these, we choose the parameter combination that minimizes the cross-validation error in the “inner” loop. With this best parameter combination, we can build a model on all training data, and evaluate it on the test data in the “outer” loop.

INSERT Figure (5) about here

Nested cross-validation is necessary because a tuning of model parameters in order to minimize the error in the outer (evaluation) loop clearly leads to biased models: The overall accuracy seems better than it actually is, since the test data has been used (if indirectly) to choose optimize model parameters.

When using plain grid search, computation time can be saved by starting with a very coarse grid, and refine the search in a second or even third round of depth search. For example, for the parameter C in SVR the initial grid could be $\{10^{-2}, 10^{-1}, 10^0, 10^1, 10^2\}$. If performance in the model selection loop turns out to be optimal for $C = 10^{-1}$, we would continue depth search with a new grid for C around the optimal value, that is $\{10^{-1.5}, 10^{-1}, 10^{-0.5}\}$. We would like to emphasize again that looking for good parameter settings should only be performed in the inner (model selection) loop. When done in the outer (evaluation), results will most likely be biased.

From our experience, we would suggest to tune in particular the kernel parameter(s) of non-linear SVR and the regularization constant C very carefully, preferably with depth search, or a fine grid right from the start. The parameters of decision trees are not that sensitive, good performance can be achieved over quite wide parameter ranges.

Grid search, as outlined above, is certainly not the only search technique that can be used here. Yet, due to its generality, it can be used to set parameters in virtually any modelling technique. Among the methods tailored to specific machine learning methods, we would like to mention in particular the approach of Wang *et al.* [14] to compute a “path” of SVR solutions when varying ϵ and C . To our knowledge, this method is not yet implemented in available SVR toolboxes.

5.1.2 Bayesian Model Selection

For Gaussian process regression, the problem of model selection can be solved in a completely different way. Here, we also have a variety of parameters to choose, such as the assumed measurement noise σ for Eq. (6), and the parameters of the covariance function, Eq. (9). The Bayesian framework allows us to define a model selection criterion called the evidence (marginal likelihood). The evidence is obtained by integrating out everything we don't know, namely all the true functional values $f(\mathbf{x}_*)$. This turns out to be [22,23]

$$L = -\frac{1}{2} \log \det(K_\theta + \sigma^2 I) - \frac{1}{2} \mathbf{y}^T (K_\theta + \sigma^2 I)^{-1} \mathbf{y} - \frac{n}{2} \log 2\pi \quad (11)$$

Here, \det denotes the determinant of a matrix. We use K_θ to explicitly denote the dependence of the covariance matrix K on a set of parameters θ of the covariance function. For example, in the case of Eq. (9), $\theta = \{s, v, u_1, \dots, u_d\}$ for a total of d descriptors.

L represents the quality of model fit with a specific parameter setting, and is a proxy for the generalization performance that was computed via cross-validation for the non-Bayesian methods in Sec. 5.1.1. Gradient ascent methods can now be used to maximize L with respect to covariance function parameters θ and the measurement variance σ^2 . We use the Broyden-Fletcher-Goldfarb-Shanno [50] method as a maximizer. See [23,51] for further details and a discussion of problems such as multi-modality.

It should be noted that maximizing the marginal likelihood is not the optimal way of selecting model parameters (the optimal way would be via full integration over all model parameters). Yet, in the case of GP models, it has been found a quite reliable and, in most cases, successful way of doing it.

The use of the Bayesian model selection criterion alleviates the user from having to perform nested cross-validation and grid search: For a given training set, the model selection criterion readily provides the optimal parameters. As a further advantage, one can easily extend a Gaussian process model by further parameters, if necessary. For example, in our work on solubility prediction [22] we used different values for the measurement noise, depending on whether one or multiple measurements were available. The appropriate values can as well be found by maximizing the marginal likelihood. As an extreme example, one can even assign weights to every descriptor (the weights u_i in Eq. (9)) and obtain, as an extra result of the model selection procedure, a ranking of descriptors in terms of their importance for predicting the property. This defines a property-specific measure of distance between compounds, and can also be used later to interpret the learned model (see Sec. 6.2).

5.2 Training and Validation

In the pharmaceutical industry, successful initial experiments to estimate the model performance are typically followed by training the model on all available data and applying it in practice. As long as new compounds stem from series that are well represented in the training set of the model, the performance will be comparable to the results from the first experiments. When new series of compounds are investigated, care has to be taken: Only when the initial performance estimation is done very carefully, one can expect to correctly estimate how well the model will perform in daily use.

Common pitfalls in model evaluation are illustrated in Fig. (6). Using *space filling algorithms* [52,53] to separate available compounds into training and test set results in the best possible overlap between the sets. Consequently, high model performance will be observed. This situation is shown in Fig. (6), left, for a simplified two dimensional descriptor space.

INSERT Figure (6) about here

When new series are investigated, we typically observe a large distance between compounds from the new series and its closest neighbor in the model's training data. In addition to the larger distance, the model often has to extrapolate, rather than interpolate. This scenario is exemplified in Fig. (6), right. Depending on the learning algorithm employed, the model can be able to extrapolate to some extent, and deliver correct results for some compounds in the new series. In such a case a reliable estimate of individual prediction uncertainty is extremely useful, because it allows to distinguish between confident predictions and wild guesses. When

evaluating the overall performance on the new series, we must expect a drastic drop in performance--in particular when compared with the optimistic estimates derived from space filling algorithms.

In conventional “leave $n\%$ out” *cross-validation*, compounds to leave out of the training set for subsequent evaluation of the model are chosen at random. This procedure is typically repeated a number of times with different random splits of the data. If most series in the training set contain, *e.g.*, at least 10 compounds and we employ “leave 50 %” out cross-validation, most series will be well represented in the respective training sets of the different random splits of the data. Only limited extrapolation will be necessary and relatively high model performance will be observed. This effect will be more pronounced the less data we leave out in training, *i.e.*, for “leave 10 % out” cross-validation, our estimated performance can, depending on the distribution of the data, be as optimistic as expected from using space filling algorithms.

The ideal way of getting a realistic estimate of the model performance is to evaluate the model in the actual day to day application scenario. This means that typically a chronological evaluation will be applied: The oldest measurements will be used for model building, the youngest measurements for evaluation. Clearly, the date of the laboratory measurement must be available. Alternatively, model building and evaluation can be performed by different teams, where the modeling team does not have access to the experimental endpoints for the youngest measurements. Such a *blind test* setup was used for the evaluation of models for aqueous solubility in [22,44], $\log D$ [43,24], $\log P$, drug likeness [54] and metabolic stability [55].

Performance on new series can also be estimated without using the time information, by a “leave one cluster out” strategy: In a first step, compounds need to be clustered in a meaningful way, for example based on structural classes or on meta-data like the project a compound was investigated in. In the second step, a model is built on the training data with one cluster removed, and evaluated on the held out cluster. This second step is repeated until each cluster has been left out once, so that predictions for all compounds are generated. If clusters tend to be small or if their size varies considerably, it may be helpful to merge small clusters before applying the algorithm.

5.3 Performance Measures

After having completed one or multiple runs of cross-validation, a “leave one cluster out” evaluation (see Sec. 5.2), or a prediction on an external validation set, an overall performance measure has to be computed. Assume we have a set of M test compounds $\mathbf{x}_{*1}, \dots, \mathbf{x}_{*M}$, with experimental values y_{*1}, \dots, y_{*M} , and model predictions $f(\mathbf{x}_{*1}), \dots, f(\mathbf{x}_{*M})$. (Note that this set can also be the model predictions on the training data in cross-validation, since one run of cross-validation generates predictions for the whole training data). For regression, two quantities seem most appropriate to measure the performance, the mean absolute error, MAE, and the mean squared error, MSE,

$$\text{MAE} = \frac{1}{M} \sum_{i=1}^M |y_{*i} - f(\mathbf{x}_{*i})| \quad (12)$$

$$\text{MSE} = \frac{1}{M} \sum_{i=1}^M (y_{*i} - f(\mathbf{x}_{*i}))^2 \quad (13)$$

Another widely used error measure is the correlation coefficient. With the average prediction $m_f = \frac{1}{M} \sum_{i=1}^M f(\mathbf{x}_{*i})$ and the average experimental value $m_y = \frac{1}{M} \sum_{i=1}^M y_{*i}$, the correlation coefficient is

$$r = \frac{\sum_{i=1}^M (f(\mathbf{x}_{*i}) - m_f)(y_{*i} - m_y)}{\sqrt{\sum_{i=1}^M (f(\mathbf{x}_{*i}) - m_f)^2 \sum_{i=1}^M (y_{*i} - m_y)^2}} \quad (14)$$

A performance measure closely related to the mean squared error is q^2 , defined as

$$q^2 = 1 - \frac{\sum_{i=1}^M (y_{*i} - f(\mathbf{x}_{*i}))^2}{\sum_{i=1}^M (y_{*i} - m_y)^2} \quad (15)$$

MAE and MSE (or its square root, the RMSE) are the performance measures that are traditionally used in the machine learning community. In the literature on QSAR modelling, r (or r^2) are used more often.

Obviously, MAE and MSE become 0 when a model can predict the experimental value perfectly, that is, predicted value $f(\mathbf{x}_{*i})$ is equal to y_{*i} . MSE is slightly more susceptible to outliers, thus a model with generally low prediction error but one or two large mis-predictions can have a larger MSE than a model with mostly predictions of mediocre accuracy.

q^2 is basically the MSE, scaled by the variance of the experimental values. The variance term in the denominator can be interpreted as the mean squared error of the most trivial model, namely one that constantly predicts the average experimental value m_y . Thus, q^2 compares the predictive model f against the most trivial model. q^2 is sometimes also called “explained variance”. A value of 0.5 means that the model can explain (predict) half of the overall variability of the data. q^2 becomes 1 when all the predicted values $f(\mathbf{x}_{*i})$ are equal to y_{*i} .

The correlation coefficient r behaves quite differently from the other performance measures. r becomes 1 if model prediction and experimental value have an *exactly linear relationship*. However, this does not mean that model prediction and experimental value match. For example, if (1,2,3,4) is the set of experimental values, and (2,4,6,8) is the set of according predictions, we obtain $r = 1$. Thus, high correlation coefficient can only be used as an indicator for model predictivity when combined with other performance measures [56].

Note that $q^2 = 1$, MAE = 0 and MSE = 0 if $f(\mathbf{x}_{*i})$ is equal to y_{*i} for all compounds, and *vice versa*. This relation does not hold for r , here we can only state: If $f(\mathbf{x}_{*i})$ is equal to y_{*i} for all compounds, then $r = 1$. The converse is not true: Knowing that $r = 1$ does not tell us how close $f(\mathbf{x}_{*i})$ and y_{*i} are (we only know that they are linearly related). Thus, when plotting q^2 versus r , no clear relationship between these quantities can be found [56].

Visual Inspection

Besides the numeric performance measures, we would like to emphasize that the most helpful tool to assess the performance of a regression model is also the most simple one: visual inspection. A simple plot of predicted versus experimental endpoint can reveal a number of valuable information. It can, for example, show that all highly soluble compounds are underestimated, or that a cluster of compounds with similar solubility always gives outlier predictions. All numeric criteria to summarize the cloud of points in such a plot can only focus on one specific aspect of the cloud, and will thus fail to fully describe it.

5.4 Evaluating Error Bars

Meaningful estimates of the domain of applicability are now being considered a key requirement for every predictive model that is used for virtual screening. Thus, also the domain of applicability (DOA) estimates or error bars need to be evaluated with the same care as the plain classifier performance.

For this evaluation, it is important to keep the semantics of the DOA estimate in mind. If the DOA estimate is meant to describe the standard deviation of the actual error, this is precisely what needs to be checked. One way of assessing the quality of error bar estimates is thus to compare the predicted and the actual distribution of errors. (Such a comparison can also be made for classification models, where we compare the predicted and the empirical probability for being active in a so-called calibration curve [57].) We will subsequently only focus on the specific case where the model predicts a Gaussian distribution, given by its mean and standard deviation. This is the case for the GP model described in Sec. 2.2, but can be achieved as well by, for example, an SVR model and a distance based heuristic to map from distance to error standard deviation.

We assess the performance of the classifier on a set of test data, with compounds $\mathbf{x}_{*1}, \dots, \mathbf{x}_{*M}$ and according experimental values y_{*1}, \dots, y_{*M} . The model we wish to evaluate outputs for each of these compounds a predicted mean $\bar{f}(\mathbf{x}_{*i})$ and a standard deviation $\text{std } f(\mathbf{x}_{*i})$. To evaluate the predicted confidences, we proceed as follows. For each compound in the test, we use the inverse cumulative density function (quantile function) F^{-1} of the standardized Gaussian distribution to compute confidence intervals of the form “True value is within the interval $\bar{f}(\mathbf{x}_{*i}) \pm v$ with confidence h per-cent”:

$$v = \text{std } f(\mathbf{x}_{*i}) F^{-1}(0.5 + h/200) \quad (16)$$

with $F^{-1}(p) = \sqrt{2} \text{erf}^{-1}(2p - 1)$. erf^{-1} is the inverse error function, which is available as a primitive on most numeric systems. With v computed, we can subsequently count the percentage of points in the validation set where the true value lies within this interval. Ideally, for h per-cent of the validation data, the experimental value should fall into the predicted h per-cent confidence interval. This check is made for several values for h .

Fig. (7) shows examples of these curves for different kinds of model on a small toy data set. From the shape of the curves, we can judge whether the model is over- or underconfident, or even identify cases where the modeling assumption is wrong. In such cases, the distribution of errors does not match the predicted distribution (for example, the predictive distribution is Gaussian, but the actual errors are a more heavy-tailed distribution).

INSERT Figure (7) about here

Note also that the above described approach of evaluating the error bars focusses only on the shape of the predictive distribution. It is to a certain degree insensitive with to the actual size of the error bar. Thus, it may be beneficial to combine it with a criterion that is sensitive to the size of the error bars, but (to a certain degree) insensitive to the shape of the predictive distribution. Such a criterion is the average log probability of the predictive distribution,

$$LP = -\frac{1}{2M} \sum_{i=1}^M \left[\log(2\pi(\text{std } f(\mathbf{x}_{*i}))^2) + \frac{(y_{*i} - \bar{f}(\mathbf{x}_{*i}))^2}{(\text{std } f(\mathbf{x}_{*i}))^2} \right] \quad (17)$$

The expression in the sum stems from evaluating a Gaussian probability density function, with mean and standard deviation given by the model's prediction, on the point of the actual experimental outcome y_{*i} . We found that LP tends to favor slightly overconfident models. For example, for the data used to generate Fig. (7), log probability would choose the overconfident model as the best one.

6 Practical Aspects

In the subsequent section, we are trying to summarize the practical aspects of the methods discussed in Sec. 2. With a careful tuning of model parameters, as described in Sec. 5, one can (in most cases) expect comparable accuracy. The choice of methods will thus also be guided by other aspects, such as: are error bars required? How large is the data set? Following a very brief summary of arguments for and against each method, we will also point to software implementations, and discuss the issue of interpretability of the methods in Sec. 6.2.

SVR

- Pro: A number of implementations available, many of them as open source software. Some implementations can handle very large data sets. The model of measurement noise used in SVR is relatively robust to outliers in the data.
- Con: Does not provide error bars. Parameter selection needs to be done by cross-validation and grid search, thus only suitable for kernel functions with few parameters.

GP

- Pro: Provides error bars. Automatic model selection possible, even for several hundreds of model parameters. Can provide a model-based descriptor relevance ranking.
- Con: Direct implementation only suitable for a maximum of around 5,000 data points (extending GP models to larger data sets is clearly possible[25] but complicates the implementation). Assumes Gaussian measurement noise, thus can be problematic with outliers. Only few implementations available.

Decision Trees

- Pro: A number of implementations available. Suitable for large data sets. Provides error bars when used in an ensemble approach (random forests).
- Con: Parameter selection needs to be done by nested cross-validation (but parameters do not need particularly fine tuning)

6.1 Software Tools

For SVMs, a large number of implementations is available. Amongst the most popular and powerful ones are

- SVM^{light}: <http://svmlight.joachims.org>, available free of charge only for non-commercial use.
- GPDT-SVM: <http://www.dm.unife.it/gpdt/>
- libsvm: <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>

All of the above tools are implemented in C, and come as command line tools for different platforms. The Shogun SVM toolbox, <http://www.shogun-toolbox.org/> is a wrapper toolbox that contains both SVM^{light} and libsvm, with powerful kernel functions and an interface to different programming languages (Matlab, R, Octave, python). A number of cheminformatics modeling programs already include SVM methods. Examples are Equibits Foresight, SimulationsPlus AdmetModeler, and others.

Despite being an older technique than SVM, fewer implementations are available for decision trees resp. random forests. Among the few are C4.5 (<http://www.rulequest.com/Personal/>), decision trees for the free statistics package R (<http://www.r-project.org/>), and the WEKA machine learning toolbox (<http://www.cs.waikato.ac.nz/ml/weka/>). The most recent implementation is JBoost (<http://www.cs.ucsd.edu/users/aarvey/jboost/>), which combines boosting with a specific variant of decision trees. The software suite PipelinePilot includes decision tree models.

For implementations of Gaussian process models, we also see a mixed picture.

- Rasmussen and Williams provide software for GP regression and classification to accompany their book [23], <http://www.gaussianprocess.org/gpml> (available only for the commercial software package Matlab)
- SPGP: <http://www.gatsby.ucl.ac.uk/~snelson/>, GP regression for more than 5,000 training data (available only for Matlab)
- Tpros: <http://wol.ra.phy.cam.ac.uk/mng10/GP/GP.html>, a C implementation of GP regression models
- The FBM Flexible Bayesian Modeling program by Radford Neal, written in C, also contains GP models, yet with a fully Bayesian treatment via MCMC sampling methods (<http://www.cs.toronto.edu/~radford/fbm.software.html>).

Following our first publication on modeling solubility with GP models [22], there has been increased interest in GP modeling. GP modeling functionality has been recently incorporated into the Admensa Interactive software [58], <http://www.biofocusdpi.com>. The GP modeling software we use in our work can be purchased from idalab, <http://www.idalab.com>.

6.2 Interpretation

Finally, we would like to briefly comment on the issue of interpretability of the discussed models.

It should be noted first that, in general, interpretation of nonlinear models is still a mostly open research topic. A prominent exception from this rule are decision trees: Each prediction can be traced fully through the tree. However, as already mentioned in Sec. 2.3, the performance (and stability) of single trees is often not sufficient. When using an ensemble of trees, direct interpretability is lost. An indirect interpretation could be gained from counting how often a specific descriptors was used in the ensemble: Higher frequency can be interpreted as high descriptor relevance.

Decision trees/forests are examples of parametric machine learning methods, where all information about the data is absorbed into the parameters of the model (in this case, the tree structure). On the other hand, GP models are an example of a non-parametric method. The prediction of the GP model is always composed of the contribution by each training compound, as can be seen in Eq. (7). All the training data is, in a way, stored in the model. One way of interpreting the GP model is to consider the covariance function: The covariance function, by definition, describes the relation of two compounds. By computing the covariance function between the test point and all training data, we can identify those training data that have the highest covariance with the test point, and thus the greatest influence on the prediction. This makes sense in particular when using a covariance function that assigns different weights to descriptors, as in Eq. (9). Weights are adapted to the prediction problem at hand by the Bayesian model selection procedure, as outlined in Sec. 5.1.2. Thus, the covariance function computes a *property specific measure of similarity*.

SVR can, in principle, be interpreted in a similar way as GP models. There are two major differences here: First, the model resulting from SVR is typically sparse, see Sec. 2.1, and only contains a fraction of the training data (though this fraction can be quite high, depending on the complexity of the modelling task and the chosen parameters). Secondly, in SVR model selection, it is virtually impossible to assign weights to each descriptor in the kernel function. Thus, every descriptor contributes similarly to the kernel function, and the kernel function will merely represent a transformed Euclidean distance (the main component of a kernel function such as Eq. (4)), without any property specific adaptation.

However, it must be emphasized that the ideas we have outlined above have not (yet) been thoroughly tested or implemented. They are merely ideas that we intend to follow in our future research.

7 Modeling Examples

We have employed the algorithms described in this paper in a number of studies to predict physico-chemical properties [22,24,44,43] as well as biological properties (*in vitro* metabolic stability, [55]). In all cases, the goal was to build models based on data provided by Bayer Schering Pharma AG. After the model building stage, models were evaluated in a blind test, using measurements of new in-house compounds of Bayer Schering Pharma. The problems

we investigated differ with respect to the amount of data (hundreds, thousands or ten thousands of training compounds) and difficulty of the problem.

As a first endpoint, we modeled aqueous solubility, based on several hundred measurements of in-house compounds of Bayer Schering Pharma and several thousand values from the literature [22,44]. Tab. (1) contains the results achieved with the methods discussed in the present paper (the results are taken from [44]). When comparing the results achieved in cross-validation and in the blind test, we observed a slight decrease of accuracy. The blind test data set contained a number of compounds that are dissimilar to the training data. By additionally taking the measure for the domain of applicability into account, as shown in the scatter-plot in Fig. (8), one can focus on the predictions inside the DOA and achieve an accuracy that is comparable to that previously estimated in cross-validation.

INSERT Figure (8) about here

Based on approx. 20,000 training compounds, we also constructed models for HPLC measurements of $\log P$ and $\log D_7$ [24,43]. Consistent with previous results [59], we found that linear models work relatively well for predicting $\log P$ and even for the inherently more complex $\log D_7$. A scatter-plot for the prediction of the GP model on a set of 7,000 blind test compounds is shown in Fig. (9), along with the curves that compare the distribution of predicted and actual error (as described in Sec. 5.4).

INSERT Figure (9) about here

In all the problems we studied, we found that, in general, both model based error bars (*i.e.*, from Bayesian and ensemble models) and the distance to the training data can be used to judge whether compounds are inside the model's domain of applicability. When focusing on compounds inside the domain of applicability, performance clearly increases.

Obrezanova *et al.* [58] also recently presented a number of modeling results with Gaussian processes. They consider GP models for a number of endpoints, including blood-brain barrier penetration, hERG inhibition, and aqueous solubility.

Summary

The research in the field of machine learning and statistical modeling has certainly lead to an impressive number of approaches that can be used as accurate tools for virtual screening. In this paper we reviewed three approaches that, as we believe, are the right choice in a number of different application scenarios. As a first method, we considered support vector regression (SVR), a method that seems particularly attractive for its handling of large data sets of $> 100,000$ training data. As the second method, we were looking at Gaussian process regression (GPR). Here, the Bayesian framework allows us to readily obtain error bars for each prediction, perform parameter selection in a fully automatic fashion, and obtain a ranking of descriptor with respect to their relevance for the prediction task. As the third method, decision trees or forests seem particularly valuable in terms of their easy interpretation.

Experience has shown that high performance can be achieved with a number of different methods, given that each method is carefully tuned. Thus, the above mentioned aspects might be in fact the main criteria in choosing the appropriate method.

Clearly, many machine learning methods can and need to be adapted to the specific problems that arise in virtual screening and related application areas. A particular important area for improvement is the issue of interpretability. Here, in a first step, well-founded mathematical approaches need to be developed in order to establish the influence of specific atoms or groups for the model's prediction. These then need to be subsequently carefully validated and checked for their chemical plausibility and applicability, before machine learning can be safely used to explain chemical or biological activity.

Acknowledgments

The authors gratefully acknowledge partial support from the PASCAL Network of Excellence (EU #506778), and from DFG grant MU 987/4-1. We thank the anonymous reviewers and the editor for their detailed comments, and Bernhard Schölkopf for the discussion on regularization in the SVR framework.

Bibliography

- [1] Blundell, T. L.; Sibanda, B. L.; Sternberg, M. J. E.; Thornton, J. M. *Nature*, **1987**, *326*, 347-352.
- [2] Netzeva, T. I. *et al. Altern. Lab. Anim.*, **2005**, *33*, 1-19.
- [3] Tetko, I. V.; Bruneau, P.; Mewes, H.-W.; Rohrer, D. C.; Poda, G. I. *Drug Discovery Today*, **2006**, *11*, 700-707.
- [4] Goldman, B. B.; Walters, W. P. In *Annual Reports in Computational Chemistry*, Spellmeyer, D. C., Ed.; Elsevier, **2006**; Vol. 2, Chapter 8, pp 127-140.
- [5] Vapnik, V. *The Nature of Statistical Learning Theory*, Springer Verlag: New York, **1995**.
- [6] O'Hagan, A. *J. Roy. Stat. Soc. B*, **1978**, *40*, 1-42.
- [7] Breiman, L.; Friedman, J.; Olshen, J.; Stone, C. *Classification and Regression Trees*, Wadsworth, **1984**.
- [8] Schölkopf, B.; Smola, A. J. *Learning with Kernels*, MIT Press: Cambridge, MA, **2002**.
- [9] Smola, A. J.; Schölkopf, B. A Tutorial on Support Vector Regression; Technical Report NEUROCOLT NC2-TR-1998-030, **1998**.
- [10] Smola, A. J.; Schölkopf, B. *Stat. Comp.*, **2004**, *14*, 199-222.
- [12] Laskov, P. In *NIPS 12*; Solla, S.; Leen, T.; Müller, K.-R., Eds.; MIT Press, **2000**; pp 484-490.
- [13] Collobert, R.; Bengio, S. *J. Mach. Learn. Res.*, **2001**, *1*, 143-160.
- [14] Wang, G.; Yeung, D.-Y.; Lochoovsky, F. H. In *ICML2006*, De Raedt, L.; Wrobel, S., Eds.; ACM Press, **2006**; pp 993-1000.
- [15] Sonnenburg, S.; Rätsch, G.; Rieck, K. In *Large Scale Kernel Machines*, Bottou, L.; Chapelle, O.; DeCoste, D.; Weston, J., Eds.; MIT Press, **2007**; pp 73-103.
- [16] Odone, F.; Barla, A.; Verri, A. *IEEE Trans. Imag. Proc.*, **2005**, *14*, 169-180.
- [17] Ralaivola, L.; Swamidass, S.; Saigo, H.; Baldi, P. *Neural Networks*, **2005**, *18*, 1093-1110.
- [18] Gärtner, T.; Lloyd, J.; Flach, P. *Mach. Learn.*, **2004**, *57*, 205-232.
- [19] Jebara, T.; Kondor, R.; Howard, A. *J. Mach. Learn. Res.*, **2004**, *5*, 819-844.
- [20] Leslie, C.; Eskin, E.; Cohen, A.; Weston, J.; Noble, W. *Bioinformatics*, **2003**, *1*, 1-10.
- [21] Vishwanathan, S.; Smola, A. In *Kernels and Bioinformatics*, Tsuda, K.; Schölkopf, B.; Vert, J., Eds.; MIT Press, **2004**; pp 113-130.
- [22] Schwaighofer, A.; Schroeter, T.; Mika, S.; Laub, J.; ter Laak, A.; Sülzle, D.; Ganzer, U.; Heinrich, N.; Müller, K.-R. *J. Chem. Inf. Model.*, **2007**, *47*, 407-424.

- [23] Rasmussen, C. E.; Williams, C. K. *Gaussian Processes for Machine Learning*, MIT Press, **2005**.
- [24] Schroeter, T.; Schwaighofer, A.; Mika, S.; ter Laak, A.; Sülzle, D.; Ganzer, U.; Heinrich, N.; Müller, K.-R. *ChemMedChem*, **2007**, *2*, 1265-1267.
- [25] Quiñonero-Candela, J.; Rasmussen, C. E. *J. Mach. Learn. Res.*, **2005**, *6*, 1939-1959.
- [26] Duda, R.; Hart, P.E.; Stork, D.G., *Pattern classification*, John Wiley & Sons, second ed.; **2001**.
- [27] Hastie, T.; Tibshirani, R.; Friedman, J. *The Elements of Statistical Learning*, Springer: New York, N.Y., **2001**.
- [28] Mallows, C. *Technometrics*, **1973**, *15*, 661-675.
- [29] Breiman, L. *Mach. Learn.*, **1996**, *26*, 123-140.
- [30] Freund, Y.; Schapire, R. E. *J. Comp. Syst. Sci.*, **1997**, *55*, 119-139.
- [31] Amit, Y.; Geman, D. *Neur. Comp.*, **1997**, *9*, 1545-1588.
- [32] Breiman, L. *Mach. Learn.*, **2001**, *45*, 5-32.
- [33] Tropsha, A. In *Annual Reports in Computational Chemistry*, Spellmeyer, D. C., Ed.; Elsevier, **2006**; Vol. 2, Chapter 7, pp 113-126.
- [34] Tong, W.; Xie, Q.; Huixiao Hong,.; Shi, L.; Fang, H.; Perkins, R. *Env. Health Persp.*, **2004**, *112*, 1249-1254.
- [35] Kühne, R.; Ebert, R.-U.; Schüürmann, G. *J. Chem. Inf. Model.*, **2006**, *46*, 636-641.
- [36] Silverman, B. W. *Density Estimation for Statistics and Data Analysis*, Chapman & Hall, **1986**.
- [37] Sheridan, R. P.; Feuston, B. P.; Maiorov, V. N.; Kearsley, S. K. *J. Chem. Inf. Model.*, **2004**, *44*, 1912-1928.
- [38] Bruneau, P.; McElroy, N. R. *J. Chem. Inf. Model.*, **2006**, *46*, 1379-1387.
- [39] Göller, A. H.; Hennemann, M.; Keldenich, J.; Clark, T. *J. Chem. Inf. Model.*, **2006**, *46*, 648-658.
- [40] Manallack, D. T.; Tehan, B. G.; Gancia, E.; Hudson, B. D.; Ford, M. G.; Livingstone, D. J.; Whitley, D. C.; Pitt, W. R. *J. Chem. Inf. Model.*, **2003**, *43*, 674-679.
- [41] Gelman, A.; Carlin, J. B.; Stern, H. S.; Rubin, D. B. *Bayesian Data Analysis*; Chapman & Hall/CRC, second ed., **2003**.
- [42] Bernardo, J. S.; Smith, A. F. M. *Bayesian Theory*; John Wiley & Sons Ltd., **1993**.
- [43] Schroeter, T.; Schwaighofer, A.; Mika, S.; Laak, A. T.; Suelzle, D.; Ganzer, U.; Heinrich, N.; Müller, K.-R. *Mol. Pharm.*, **2007**, *4*, 524-538.
- [44] Schroeter, T.; Schwaighofer, A.; Mika, S.; Laak, A. T.; Suelzle, D.; Ganzer, U.; Heinrich, N.; Müller, K.-R. *J. Comput.-Aided Mol. Des.*, **2007**, online, DOI 10.1007/s10822-007-9125-z
- [45] Todeschini, R.; Consonni, V. *Handbook of Molecular Descriptors*; John Wiley & Sons, Ltd., **2000**.
- [46] Kless, A.; Eitrich, T. In *Knowledge Exploration in Life Science Informatics: International Symposium KELSI 2004*, Springer Verlag, **2004**; Vol. 3303, pp 191-205.
- [47] Wegner, J. K.; Zell, A. *J. Chem. Inf. Comput. Sci.*, **2003**, *43*, 1077-1084.
- [48] Liu, Y. *J. Chem. Inf. Comput. Sci.*, **2004**, *44*, 1823-1828.
- [49] Wegner, J. K.; Fröhlich, H.; Zell, A. *J. Chem. Inf. Comput. Sci.*, **2004**, *44*, 931-939.
- [50] Zhu, C.; Byrd, R. H.; Nocedal, J. *ACM Trans. Math. Software*, **1997**, *23*, 550-560.
- [51] Neal, R. M. In *Bayesian Statistics 6*, Bernardo, J. M.; Berger, J.; Dawid, A.; Smith, A., Eds.; Oxford University Press, **1998**; Vol. 6, pp 475-501.
- [52] Yan, A.; Gasteiger, J. *QSAR Comb. Sci.*, **2003**, *22*, 821-829.
- [53] Yan, A.; Gasteiger, J.; Krug, M.; Anzali, S. *J. Comput.-Aided Mol. Des.*, **2004**, *18*, 75-87.
- [54] Müller, K.-R.; Rättsch, G.; Sonnenburg, S.; Mika, S.; Grimm, M.; Heinrich, N. *J. Chem. Inf. Model*, **2005**, *45*, 249-253.

- [55] Schwaighofer, A.; Schroeter, T.; Mika, S.; ter Laak, A.; Lienau, P.; Reichel, A.; Heinrich, N.; Müller, K.-R. *J. Chem. Inf. Model.*, submitted.
- [56] Golbraikh, A.; Tropsha, A. *J. Mol. Graphics Modell.*, **2002**, *20*, 269-276.
- [57] Murphy, A. H.; Winkler, R. L. *Int. J. Forecasting*, **1992**, *7*, 435-455.
- [58] Obrezanova, O.; Csányi, G.; Gola, J. M.; Segall, M. D. *J. Chem. Inf. Model.*, **2007**, *47*, 1847-1857.
- [59] Ghose, A. K.; Crippen, G. M. *J. Comput. Chem.*, **1986**, *7*, 565-577.
- [60] Hong, H.; Tong, W.; Fang, H.; Shi, L.; Xie, Q.; Wu, J.; Perkins, R.; Walker, J. D.; Branham, W.; Sheehan, D. M. *Env. Health Persp.*, **2002**, *110*, 29-36.

Figures

INSERT schwaighofer_figure1.pdf here, width approx 6cm

Figure 1: An illustration of support vector regression (SVR) on a simple one-dimensional data set. When fitting an SVR model to data, errors that are smaller than ϵ will be ignored. Errors larger than ϵ contribute linearly to the objective function in Eq. (2) via the variable ξ_i .

INSERT
schwaighofer_figure2a.pdf
here, width approx 4cm
(a) *a priori*

INSERT
schwaighofer_figure2b.pdf
here, width approx 4cm
(b) after observing data

INSERT
schwaighofer_figure2c.pdf
here, width approx 4cm
(c) predictive distribution

Figure 2: Bayesian modeling with Gaussian processes. See Sec. 2.2 for a detailed description of the graphs.

INSERT schwaighofer_figure3.pdf here, width approx 10cm

Figure 3: An example for a decision tree that classifies compounds as active (A) and inactive (I) with respect to estrogen receptor binding (taken from [60])

INSERT
schwaighofer_figure4a.pdf
here, width approx 6cm
(a) Support vector machine
with distance based error
estimation

INSERT
schwaighofer_figure4b.pdf
here, width approx 6cm
(b) Random forest with
ensemble based error
estimation

INSERT
schwaighofer_figure4c.pdf
here, width approx 6cm
(c) Gaussian process with
Bayesian error estimation

Figure 4: The different regression models employed in this study are trained on a small number of noisy measurements (black crosses) produced from the sine function (grey dashed line). Predictions from each model are drawn as solid black lines, while dashed black lines indicate errors estimated by the respective model (for Gaussian process and random forest models) or a distance based approach (for support vector regression).

INSERT schwaighofer_figure5.pdf here, approx width 7cm

Figure 5: Nested cross-validation for model evaluation and parameter selection. In an outer evaluation loop, data are divided into training and test data. The training data are again split into training and test parts in the inner (model selection) loop. We choose those model parameters that maximize the performance on the test data in the inner (model selection) loop. These parameters can then be used to build a model on all training data from the outer (evaluation) loop.

INSERT

schwaighofer_figure6a.pdf
here, width approx 4cm

(a) Cross-validation

INSERT

schwaighofer_figure6b.pdf
here, width approx 4cm

(b) Leave one cluster out

INSERT

schwaighofer_figure6c.pdf
here, width approx 4cm

(c) New data

Figure 6: Cross-validation and space filling algorithms (a) tend to yield too optimistic performance estimates when compared with evaluation on new data, if new series are investigated (c). We propose a “leave one cluster out” strategy (b) to allow for realistic performance evaluations.

INSERT schwaighofer_figure7.pdf here, width approx 6cm

Figure 7: Evaluating error bars of a regression model. The plots compare the predicted and the empirical distribution of errors for an underconfident and an overconfident model. The dashed line shows a typical example where the empirical error is not Gaussian distributed (wrong model assumption)

INSERT schwaighofer_figure8.pdf here, width approx 5.5cm

Figure 8: Scatter-plot for solubility predictions by a Gaussian process model on blind test data that contains many structures that are dissimilar to the training data (taken from [44]). Black points represent confident predictions, grey points represent less confident predictions with predicted error bars larger than 0.6 log units.

INSERT schwaighofer_figure9a.pdf here, width approx 8cm

(a) Predicted versus experimental

INSERT schwaighofer_figure9b.pdf here, width approx 3.5cm

(b) Comparison of actual and predicted error distribution

Figure 9: Scatter-plot for $\log D_7$ predictions by a Gaussian process model on a blind test data set of $\approx 7,000$ molecules, taken from [24], and the evaluation of error bars as described in Sec. 5.4.

Tables

Table 1: Accuracy for predicting buffer solubility of drug discovery molecules when using Gaussian process models, support vector regression, and random forests. “Mean” denotes the results achieved when always predicting the arithmetic mean solubility of the respective dataset. Accuracy is measured in terms of MAE, Eq. (12), and RMSE, Eq. (13). For a full description and a discussion of the results, see [44]

| In-house data, cross-validation | MAE | RMSE |
|--|-------|-------|
| GP | 0.573 | 0.747 |
| SVR | 0.617 | 0.803 |
| Random Forest | 0.650 | 0.840 |
| mean | 0.908 | 1.117 |

| In-house data, blind test | MAE | RMSE |
|----------------------------------|-------|-------|
| GP | 0.656 | 0.846 |
| SVR | 0.657 | 0.848 |
| Random Forest | 0.663 | 0.855 |
| mean | 1.008 | 1.305 |