Johannes Gutenberg University Mainz

Faculty 08
Institute of Computer Science
Data Mining

Data Mining and Machine Learning

Machine Learning Based Prediction of Half-Lives of Environmental Pollutants

Steven Lang

1. Reviewer	Univ.-Prof. Dr. Stefan Kramer Data Mining
2. Reviewer	Univ.-Prof. Dr. Andreas Hildebrandt Scientific Computing and Bioinformatics
Supervisor	Dr. Jörg Wicker

September 1, 2017

Steven Lang

Machine Learning Based Prediction of Half-Lives of Environmental Pollutants

Data Mining and Machine Learning, September 1, 2017

Reviewers: Univ.-Prof. Dr. Stefan Kramer and Univ.-Prof. Dr. Andreas Hildebrandt

Supervisors: Dr. Jörg Wicker

Johannes Gutenberg University Mainz

Data Mining

Institute of Computer Science

Faculty 08

Saarstr. 21

55122 Mainz

Declaration

I hereby declare that I have written the present thesis independently and without use of other than the indicated means. I also declare that to the best of my knowledge all passages taken from published and unpublished sources have been referenced. The paper has not been submitted for evaluation to any other examining authority nor has it been published in any form whatsoever.

Mainz, September 1, 2017

Steven Lang

Abstract

Estimating degradation times of molecules, that possibly end up in the environment, is an important aspect for a wide range of industries. They have to take into account regulations that set a legal limit on half-lives of potential harmful molecules used in e.g. drugs or pesticides. This thesis addresses this by building machine learning models for the prediction of environmental pollutant's half-lives. The recently released Eawag-Soil dataset of compounds and environmental conditions is used to explore possible influences on the molecule's persistence. Moreover, a novel neural network architecture is proposed that takes advantage of the bi-relational data scheme.

The results indicate that the dataset does either not have enough environmental variables or not enough combinations of compounds with degradation scenarios to explore the influences of the environmental conditions on the half-life times.

Abstract (german)

Das Abschätzen von Abbauzeiten von Molekülen, welche möglicherweise in der Umwelt enden, ist ein wichtiger Aspekt für viele Industrien. Sie müssen auf Regularien achten, welche eine legale Grenze für Halbwertszeiten von Molekülen in Medikamenten oder Pestiziden setzen. Deshalb fokussiert sich diese Arbeit auf das erstellen von Machine Learning Modellen zur Vorhersage der Halbwertszeiten von Schadstoffen. Der kürzlich veröffentlichte Eawag-Soil Datensatz, bestehend aus chemischen Verbindungen und Umweltbedingungen, wird benutzt um mögliche Einwirkungen auf die Persistenz eines Moleküls zu untersuchen. Des Weiteren wird eine neue Neuronale Netzwerk Architektur vorgestellt, welche sich das bi-relationale Datenschema zu Nutzen macht.

Die Ergebnisse zeigen, dass der Datensatz entweder nicht genügend Umwelt-Variablen beinhaltet oder nicht genügend Kombinationen von Molekülen mit Abbau-Scenarien vorhanden sind um den Einfluss von Umweltbedingungen auf die Halbwertszeiten zu untersuchen.

Contents

1 Introduction

Each day, industries release chemicals into the environment. These can cause harm, e.g. pesticides that reach destinations other than their target species. Therefore, it is important to know the persistence of released chemicals. While it is acceptable if they are extremely toxic but their degradation time is in the scale of milliseconds, persistence is strongly undesired. Regulations require corporations to use chemicals of limited lifetime in the environment. It is therefore necessary to understand the degradation processes under certain environmental conditions and approximate the half-life, i.e. the time it requires to reduce its mass to half its initial value, of emerging pollutants.

The recent release of the Eawag-Soil package [16], containing microbial biotransformation pathways and meta-data of organic contaminants in different environments, allows an analysis of persistencies and opens up new possibilities to build models for the prediction of pollutant's half-lives.

Quantitative structure-biodegradation relationship (QSBR) models are predictors for the readiness of biodegradation, i.e., the half-life, using the compound's chemical structure. Their predictive power for binary tasks, such as whether a chemical is readily biodegradable or not, yield reasonably accurate classifications [4, 35, 20]. However, for risk assessment purposes a numeric regression on half-lives under specific environmental conditions is necessary, at which these models fall short [2, 10, 15]. Latino et al. [16] have carried out multiple linear regression on groups of similar pesticides achieving promising results on influences of single environment variables on the pesticide's persistence. This thesis takes their approach one step further and builds more advanced models for a heterogeneous set of compound groups, trying to explore more general dependencies between the environmental conditions and their half-lives.

To solve the problem of half-life estimation, a new approach is introduced: An advanced neural network architecture, that makes use of the bi-relational data scheme. It consists of two denoising autoencoder, learning a condensed representation of the data and forwarding it to a third network with the task of half-life regression. This approach is evaluated and compared to Random Forests and Support Vector Machines. For each method, parameters are explained, an experimental setup is described and evaluated on the dataset, visualizing predictions as well as parameter influences. Finally, two permutation tests on the dataset are performed to further reveal possible issues that arise with the dataset.

The rest of this thesis is organized as follows: Chapter 2 introduces the dataset extracted from the Eawag-Soil package. Chapter 3 explains the approaches used for the experiments, whose setups are defined in Chapter 4. Chapter 5 describes and evaluates the experimental results and concluding remarks are given in Chapter 6.

Data 2

The dataset was published by Latino et al. [16] and is available on *enviPath* [36] - an environmental contaminant biotransformation pathway resource system. EnviPath offers a variety of biodegradation pathways. These include compounds, reactions between compounds and information about the environmental conditions under which the reactions took place. A large set of compounds from this dataset is annotated with a half-life, given a set of environmental conditions, a so-called *scenario*. The package consists of all EU registered and freely accessible regulatory data on pesticide degradation in laboratory soil simulation studies under aerobic conditions for pesticides. Eawag-Soil contains 282 pathways, 2188 reactions, 2366 compounds and 4890 biotransformation half-lives.

The extracted data thereby consist of two relations and a target variable. The first relation is the set of compounds $\mathbf{C} = \{\mathbf{c}_1, \ldots, \mathbf{c}_m\}$ which are part of a pathway in Eawag-Soil. The second relation forms the set of scenarios $\mathbf{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$, i.e. the environmental conditions under which a compound degraded in a pathway. The target variable can be described as a matrix $H \in \mathbb{R}^{m \times n}$ where h_{ij} corresponds to the half-life of compound $\mathbf{c}_i$ under the environmental conditions of scenario $\mathbf{s}_j$. As the package provides 744 unique compounds and 3108 unique scenarios, this matrix is sparse with an occupation rate of 0.21%. This leads to a problem in exploring deeper correlations between the environment and the compound's structure, which is further discussed in Chapter 5.

For the approaches presented in Chapter 3 it is often necessary to standardize the input features. This is done by transforming each datapoint with

$$z^{(k)} = \frac{x^{(k)} - \mu^{(k)}}{\sigma^{(k)}} \quad , \tag{2.1}$$

where $x^{(k)}$ is the k-th feature value of a datapoint $\mathbf{x}$ and $\mu^{(k)}$ and $\sigma^{(k)}$ are the mean and standard deviation of the k-th feature.

2.1 Compounds

The compounds in the Eawag-Soil package are given as strings in the *Simplified molecular-input line-entry system* format *SMILES*, which allows the encoding of an arbitrary complex compound structure into a simple ASCII string. Fig. 2.1 shows the structure of (a) 7-OH-metosulam, which translates to

CC1=CC=C(C(=C1Cl)NS(=O)(=O)C2=NN3C(=CC(=NC3=N2)OC)O)Cl

in SMILES and (b) Diazepam which can be expressed as

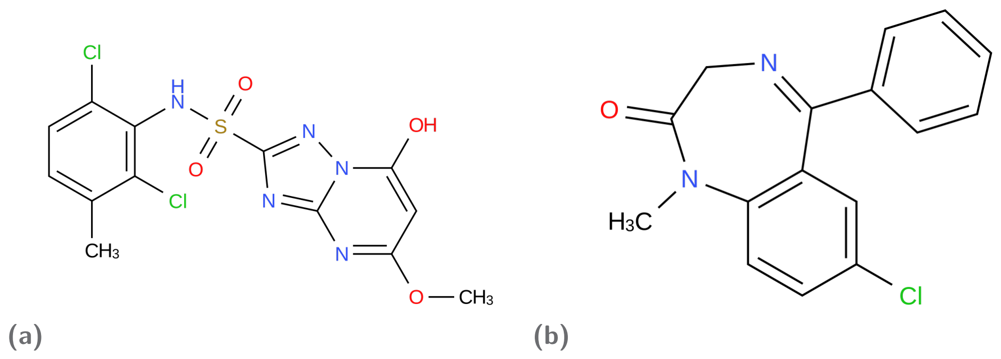


Fig. 2.1: Molecule structures of (a) 7-OH-metosulam and (b) Diazepam

CN1C2=C(C=C(C=C2)Cl)/C(=N\CC1=O)/C3=CC=CC=C3.

The data set consists of a wide range of structures of different sizes. The smallest structure is Phosphorus (*P*) whereas the biggest is build of up to 91 atoms. On the average, a compound contains 22 atoms.

Molecule Descriptors

The SMILES representation itself cannot be used as a descriptor of a compound. Since it fully encodes the molecule's structure it delivers access to the graph $G \in \mathbb{L} \times \mathbb{L} \times \mathbb{N}$, where $\mathbb{L}$ is the label set of atoms. A triple $g \in G$, with $g = (i, j, k)$ represents a k electron bond between atoms with the label i and j. This graph can now be queried by different strategies, or so-called *Fingerprinter*. In general, a Fingerprinter is a mapping $F : G \to \{0, 1\}^n$, where F contains n queries against a graph. $F(g) = \mathbf{b}$ is a bit-vector of size n and can be read as:

$\forall i \in \{1, \ldots, n\}$, b_i is the result of the i-th query of the Fingerprinter.

The most common one is the *MACCS* Fingerprinter [8], which defines 166 unique queries against the molecule. Each query contains a question about the chemical structure, such as "*Are there fewer than 3 oxygens?*" or "*Is there a ring of size 4?*". Examples are shown in Tab. 2.1. A query is assigned an index and a condition. The result of the query is 1 or $true$ if the molecule satisfies the condition and 0 or $false$ otherwise.

Another Fingerprinter is the *FP2* (1022 queries), which locates molecule fragments in linear segments of up to 7 atoms in length. *FP3* (56 queries) and *FP4* (308 queries) are Fingerprinter that create a bit-vector by predefined *SMARTS* pattern. SMARTS is an extension to SMILES, developed for the process of finding a specific substructure pattern in a molecule. All Fingerprinter named above are implemented in the open source chemical toolbox software *Open Babel* [24].

Therefore, the compound relation can be described as a matrix $C \in \{0, 1\}^{m \times n}$ where $m = 744$ is the number of unique compounds and n is the bit-vector size of

Index	Query	Result	
		7-OH-metosulam	Diazepam
11	4M RING	0	0
14	S-S	0	0
19	7M RING	0	1
45	C=CN	0	0
78	C=N	0	1
92	OC(N)C	1	1
163	6M RING	1	1

Tab. 2.1: Sampled MACCS query results for 7-OH-metosulam (Fig. 2.1a) and Diazepam (Fig. 2.1b). Each query has its Fingerprint index, a defined condition and a result of 1 if the queried molecule meets the condition or 0 if not.

Parameter	Unit	Miss.	Min.	Mean	Med.	Max.	Std. dev.
Acidity, pH	–	0.02	3.6	6.63	6.8	8.8	0.88
Biomass end	mg C g^{-1}	0.43	0.0	0.54	0.26	88.5	2.7
Biomass start	mg C g^{-1}	0.32	0.0	0.67	0.32	93.7	2.95
Bulk density	g cm^{-1}	0.62	0.0	1.33	1.36	2.66	0.29
CEC	meq./100 g soil	0.19	0.05	15.11	12.6	209.0	14.27
OC	g OC/100 g soil	0.04	0.02	1.79	1.6	10.0	1.07
Spike concentration	mg per kg dry soil	0.15	0.0	5.29	0.4	500.0	32.52
Temperature	°C	0.02	1.0	20.16	20.0	49.0	3.6
Water storage capacity	g water/100 g dry soil	0.16	1.54	39.87	38.0	816.1	21.85
% humidity	–	0.19	5.0	50.69	41.0	100.0	17.04
% clay	–	0.11	0.1	16.01	12.44	94.3	10.98
% sand	–	0.12	0.0	52.01	55.0	99.0	24.6
% silt	–	0.12	0.0	31.98	28.0	88.6	18.76

Tab. 2.2: Statistics of the environmental conditions dataset extracted from the Eawag-Soil package, listing for each parameter its unit, fraction of missing values, minimum, mean, median, maximum value and standard deviation.

the used Fingerprinter. A comparison of model influences between MACCS, FP2, FP3 and FP4 is shown in Chapter 5.

2.2 Environmental Conditions

The second relation is the set of scenarios, where each scenario describes 15 conditions under which a degradation process took place. These can be either measured parameters such as the *acidity* and *water storage capacity*, or parameters which where chosen in laboratory experiments beforehand like the *temperature* and *biomass* of the tested compounds. There are two categorical features, namely *soilclassificationsystem* – the system, the soil has been classified with – (2 values) and *soiltexture1* – the soil classification result of the system named in *soilclassificationsystem* – (12 values). The other 13 conditions are numerical. A short summary statistic is shown in Tab. 2.2.

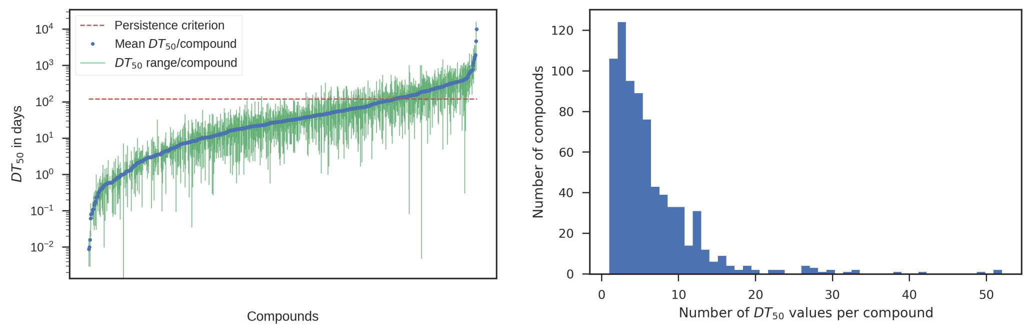
Preprocessing

The environmental conditions were manually extracted from existing dossiers. This process is error prone and introduces reading and typing mistakes which need to be curated in the data preprocessing step. Since errors cannot be fixed due to the lack of knowledge of the original parameter, one can drop certain values based on logical and physical decisions. That is, the *humidity* attribute is expressed as percentage and in eight cases its value is above 100%. The parameters *silk*, *sand* and *clay* are also given as percentage, such that they should add up to 100%. The values of *silk*, *sand*, and *clay* were dropped for a total of 51 scenarios that had a sum higher than 100%. For the organic content (*OC*) attribute, values lower than 0 and higher than 10 g OC/100 g soil were removed in 12 rows as recommended in Latino et al. [16]. Further, entries of *OC*, given in the organic matter measurement *OM*, were transformed by the relation $OC = OM/1.724$ [30]. Latino et al. [16] proposed two additional attributes, the experimental moisture content of the soil, which potentially influences the degradation process. It is obtained by multiplying the *waterstoragecapacity* with *humidity*. The second attribute is the degradation mass, i.e. the difference between the *biomassStart* and *biomassEnd* values.

Another problem that needed to be addressed in the preprocessing step is the imputation of missing values. The *acidity* and *temperature* attributes show a presence of 98% and are the parameters with the lowest fraction of missing values, whereas *bulkdensity* is only present in 38% of all scenarios. There are several approaches to accomplish this task. A simple method is the mean imputation. Here, all missing values of a variable are being replaced by its mean value, which is robust and fast in general. A more sophisticated estimate is the *KNN* imputation. To find a value for the j-th parameter in the i-th scenario s_{ij}, the k nearest neighbors of s_i are chosen based on the scenario matrix holding out the *j*-th. The missing value will then be imputed by the mean parameter value of the neighbors. These and further advanced methods such as a straight forward *Matrix Factorization*, *Spectral Regularization* [21], and *Multiple Imputation by Chained equations* [38] are tested on the dataset with regard to model performances in Chapter 5.

2.3 Half-Lives

The target value of the dataset is the half-life, or the so-called DT_{50} value, of a compound under certain environmental conditions. Out of 744 compounds, only 55 are annotated with exactly one half-life, 330 compounds have more than five half-lives and even 106 compounds have more than ten. The exact distribution of the number of half-lives per compound is shown in Fig. 2.2b. Compounds with more than five associated biodegradation values have a mean standard deviation of 65 days, suggesting a strong impact coming from scenarios. Fig. 2.2a shows the



(a) (b)

Fig. 2.2: (a) illustrates DT_{50} value ranges for each compound, sorted by their mean (blue dots). Green bars show minimum and maximum DT_{50} for a compound. Red dotted line is the 120 days persistence criterion. (b) shows the distribution of the number of half-lives per compound.

large intracompound half-life variability (green bars) for each compound, sorted by their mean value (blue dots). This fact should support the dependency exploration between experimental conditions and the degradation process. On the other hand there are 2218 scenarios attached to a single compound, 73 scenarios having more than five and only eight scenarios having more than ten compounds. This could make it more difficult for models to detect structural dependencies of different compounds, given the same scenario, on half-lives. Though the mean standard deviation of 62.8 days for scenarios with more than five compounds also suggest an equally important correlation between the compound's structure and its degradation.

Another aspect of the data set is the fraction of persistent compounds. A regulation proposed in *Regulation (EC) No. 1107/2009* of the European Parliament [9] defines a persistence criterion for pesticides and industrial chemicals in soil with half-lives above 120 days (red dotted line in Fig. 2.2a). This criterion is met by 18.7% of the DT_{50} values, which should be a proportion big enough for building models specialized on the binary task of predicting whether a compound is persistent or not. Latino et al. [16] further state, that half-lives above 1000 days (0.838%) are indicators for the presence of stable compounds. The dossiers study time, from which the data has been extracted, was limited to 120 days, implying that those values are extrapolated and should be considered only as a approximation of the real degradation time.

For the experimental results in Chapter 5, the half-lives were transformed by the natural logarithm, so that prediction errors of long half-lives do not outweight those of short ones.

Methods

3

The main goal of this thesis is to explore whether there exist dependencies between a molecules structure, a set of environmental conditions and the time it takes the degradation process to reduce the mass of the compound to the half of its original value. This can be achieved with methods of *supervised learning* in *machine learning*. The setup for machine learning applications usually consist of a dataset $D = (\mathbf{X}, \mathbf{y})$ with datapoints $\mathbf{x}_i \in \mathbf{X}$ and a corresponding target variable $y_i \in \mathbf{y}$. Machine learning algorithms aim to find a model f that approximates $f(\mathbf{x_i}) \approx y_i$ for all datapoints. This model can then be used to predict the target variable's value of unseen data. The dataset described in Chapter 2 provides the input relations of compounds $\mathbf{X_c}$ and scenarios $\mathbf{X_s}$, and the target variable of half-lives $\mathbf{y}$. The task of exploring dependencies in such data thus perfectly fits the setup of machine learning based methods.

3.1 Baseline Approaches

The methods used in this thesis are separated in established baseline methods which are presented in this section and more recently developed advanced methods, being more experimentally and relying heavily on the model-parameters chosen, introduced in following section. The baseline methods explained are Decision Trees (Section 3.1.1), Tree Ensembles (Section 3.1.2) and Support Vector Machines (Section 3.1.3).

3.1.1 Decision Tree

The first decision tree models appeared in the early 80's [27]. Their goal is to split the input data into distinct regions by defining rules, which are represented in a hierarchical decision tree as shown in Fig. 3.1. Each region is then assigned an output value. These regions are used to make a prediction of new inputs as follows: First, the root node's attribute of a new instance is queried. Based on the instance's value of this attribute, a path with the matching label is chosen. This path leads to the next attribute that is to be queried. These steps are repeated until the path ends in a leaf. In the case of a leaf, its label indicates the predicted output for the input instance. Fig. 3.1 illustrates a learned decision tree which classifies Saturday mornings depending on whether they are appropriate for playing tennis or not.

Decision trees can also be translated into a set of rules, namely a disjunction of conjunctions of tests against the instance's attribute values. The tree in Fig. 3.1 can then be written as

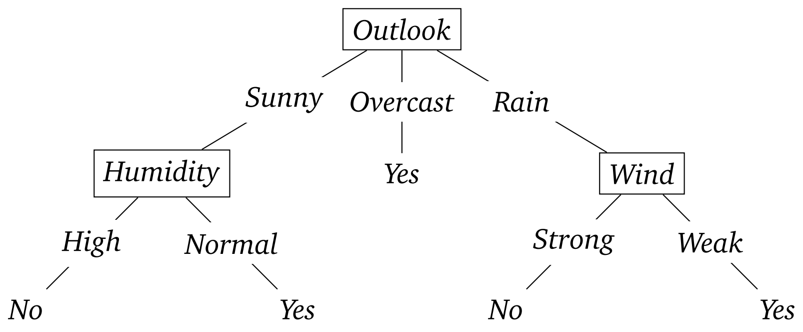


Fig. 3.1: Decision tree model for the *PlayTennis* relation as described in Mitchell [23]. Each node in the tree (squared boxes) define the test of a certain attribute. The paths between two nodes are possible manifestations of the attribute. The model classifies a new instance by starting at the root node and following the path to the leafs based on the attribute values of the instance. The leafs contain the predicted output.

$$\begin{array}{ll} & (Outlook = Sunny \wedge Humidity = Normal) \\ \vee & (Outlook = Overcast) \\ \vee & (Outlook = Rain \wedge Wind = Weak) \quad . \end{array}$$

There are multiple algorithms of decision tree based models such as ID3 (Quinlan [27]), its successor C4.5 (Quinlan [28]) and its proprietary latest improvement C5.0. For the sake of simplicity the following roughly describes the main ideas of ID3, whereas the results in Chapter 5 were produced with an enhanced version of the *CART* algorithm (Breiman [5]).

The key task of the decision tree algorithm is the choice of the order of test attributes for each node in the tree. A statistical property, *information gain*, defined as

$$Gain(S,A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{3.1}$$

quantifies how well a certain attribute *A* splits up the set of training examples S. S_v is the subset of S for which the attribute A has value v. The impurity of a set of examples S regarding the target variable is measured by the *Entropy* from information theory

$$Entropy(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad , \tag{3.2}$$

where c is the number of available classes, and p_i is the prior probability of a datapoint belonging to class i. Therefore, Eq. (3.2) measures how diverse the set of examples S is, with regard to their classes. The goal of *information gain* is to measure the expected reduction in entropy, based on the assumption that the next splitting attribute is *A*. This can be seen in Eq. (3.1), where the minuend is the entropy of the original collection and the subtrahend is the expected entropy of *S* after it is partitioned by attribute *A*. To find the best splitting attribute at each step of the tree growing phase, it only needs to be evaluated which attribute provides the most *information gain*. A node is defined as a leaf once the collection S only consists of examples of the same class, or one class has a majority of a predefined ratio p, which is commonly referred to as the *pruning threshold*.

3.1.2 Bagging and Random Forests

An important aspect of machine learning models is the *bias-variance decomposition* as described by Hastie et al. [12]. The error of a model f on test data is composed of

$$Err(f,\mathbf{X}) = IrreducableError(\mathbf{X}) + Bias(f,\mathbf{X})^2 + Variance(f,\mathbf{X}) \quad . \tag{3.3}$$

The *bias* describes the difference between the average prediction of a model and the true mean of the target variable, whereas the *variance* term quantifies the deviation of the predictions itself. As an example, a model with low complexity, e.g. a decision stump (a decision tree with only one node), most likely has a high bias and low variance due to its simple hypothesis. Growing the tree increases the model complexity and allows the bias to decrease through a more accurate separation of the input data, while the variance increases since the model can now make more diverse predictions. Usually it is necessary to find an optimum of model complexity to have a low bias but keep a low variance as well. This problem is tackled by the approach of *Bagging*, which combines *bootstrapping* and *aggregating*. The former is a way to generate additional datasets by sampling n instances from the input data with replacement. The probability of a data point not being picked is $1-\frac{1}{n}$. By sampling n-times, the probability of a data point being in the bootstrap sample is then $1-(1-\frac{1}{n})^n \approx 1-e^{-1} = 0.632$, therefore a bootstrap sample will contain approximately 63.2% of the instances. Bagging thus builds an ensemble of models, where each model is trained on a randomly bootstrapped sample of the original dataset. The actual prediction of the ensemble is generated by voting and averaging over the single predictions of the models. This procedure has shown to reduce the variance component of the expected error.

A common application of bagging is the *Random Forest* algorithm proposed by Breiman [31]. It uses decision trees as base models and combines the bagging approach with additional random feature selection. Each decision tree is trained on a bootstrapped dataset with only a fraction of the original number of features k. A good choice for k is typically $\sqrt{m}$ where m is the size of the full feature set. This is also called the *random subspace method*.

3.1.3 Support Vector Machines

Support Vector Machines have been introduced in 1995 by Cortes and Vapnik [7] even though their idea has been discussed even earlier. In the following, a brief explanation of its principals is given. To start simple, the input data is assumed to

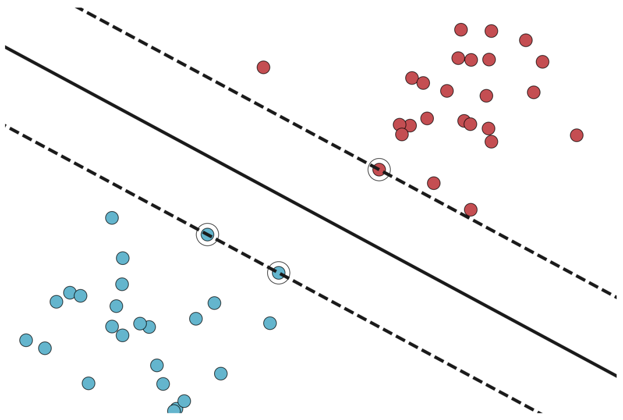


Fig. 3.2: Hyperplane (continuous line) partitioning the datapoints into positive (blue) and negative (red) examples. Support vectors are marked with circles and lie on the margin (dotted lines).

consist of two classes with $y_i \in \{-1, 1\}$ and is linearly separable. Each datapoint is seen as a vector in the input feature space. The SVM method aims to find parameters $\mathbf{w}$ and b for the linear decision function

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b \tag{3.4}$$

with the hyperplane $\mathbf{w}^T\mathbf{x} + b = 0$ that separates the datapoints

$$\mathbf{w}^T\mathbf{x}_i^+ + b \geq 1 \tag{3.5}$$

$$\mathbf{w}^T\mathbf{x}_i^- + b \leq -1 \tag{3.6}$$

such that all positive examples $\mathbf{x}_i^+$ are on the one side of the hyperplane (see Eq. (3.5)) while all negative points $\mathbf{x}_i^-$ are on the other side (see Eq. (3.6)), as shown in Fig. 3.2. The distance of a point $\mathbf{x}_i$ to the hyperplane is given by

$$\frac{y_i y(\mathbf{x}_i)}{||\mathbf{w}||} = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{||\mathbf{w}||} \tag{3.7}$$

and the margin is defined by the orthogonal distance to the closest point x_i. The goal is to optimize $\mathbf{w}$ and b such that the margin is maximized. This can be done by solving the optimization problem

$$\underset{\mathbf{w},b}{\arg\max} \left\{ \frac{1}{||\mathbf{w}||} \min_i \left[y_i \left(\mathbf{w}^T\mathbf{x}_i + b \right) \right] \right\} \tag{3.8}$$

w.r.t. $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$ as mentioned in Eq. (3.5) and (3.6).

Furthermore, SVMs allow the use of the so-called *kernel trick* for problems which are not linearly separable as assumed above, that is calculating the dot product in a feature space in which the data is linearly separable again. Common kernels are the linear kernel

$$k_{lin}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \quad , \tag{3.9}$$

the *d*-degree polynomial kernel

$$k_{poly}(\mathbf{x}_i, \mathbf{x}_j) = \left(\mathbf{x}_i^T \mathbf{x}_j + c\right)^d \tag{3.10}$$

and the radial basis kernel

$$k_{rbf}(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \quad . \tag{3.11}$$

Even though the approach described here only handles classification, SVMs can be extended to support regression as described in Bishop [3].

3.2 Advanced Methods

This chapter shall give a rough understanding of the field of *neural networks*. Even though their idea and first implementations exist since 1942 [22] as well, recent developments [29] have awakened a broader interest in both research and industrial application. Section 3.2.1 gives a brief introduction into neural networks, whereas 3.2.2 shows the concept of so-called *Autoencoder*.

3.2.1 Neural Network

A basic neural network can be described as a series of functional transformations, that is beginning with *M* linear combinations of the input $\mathbf{x} \in \mathbb{R}^D$. The output of layer (1) is defined as

$$a_j^{(1)} = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + b_j^{(1)} \tag{3.12}$$

with the *weights* w_{ji} and the *bias* b_j of the first layer and $j = 1, \dots, M$. In order to break linearity a nonlinear *activation function* $h(\cdot)$ is applied on the output of layer (1), namely $z_j^{(1)} = h(a_j^{(1)})$. Common activation functions are the *binary step* $h(x) = \max\{0, sign(x)\}$, the *tanh* function $h(x) = \tanh(x)$, and the more recently used rectified linear unit *ReLU* $h(x) = max(0, x)$ and its modifications. The *M* output values will be used as input for the next layer

$$a_k^{(2)} = \sum_{i=1}^{M} w_{ji}^{(2)} z_i^{(1)} + b_j^{(2)} \quad , \tag{3.13}$$

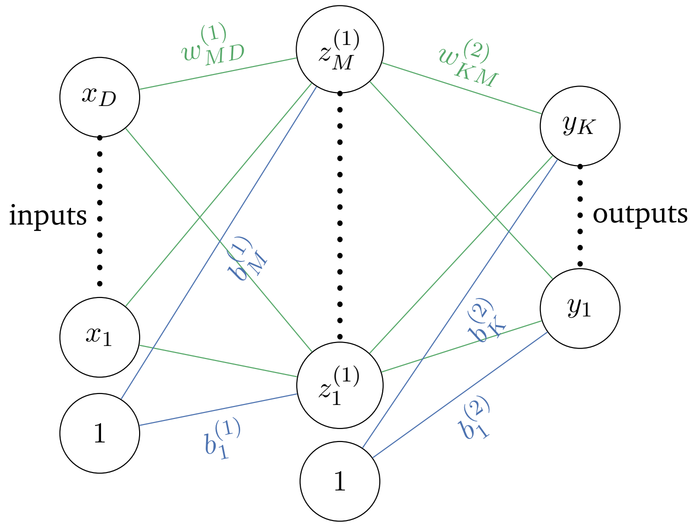


Fig. 3.3: A two layered neural network. Weights are marked green, biases are blue. The input layer takes a vector $\mathbf{x}$ of dimension D. The first layer produces the activations $\mathbf{z}^{(1)} = h^{(1)}\left(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}\right)$, with $\mathbf{W}^{(1)} \in \mathbb{R}^{M\times D}, \mathbf{b}^{(1)} \in \mathbb{R}^{M}$. The network output is generated accordingly by $\mathbf{y} = h^{(2)}\left(\mathbf{W}^{(2)}\mathbf{z}^{(1)} + \mathbf{b}^{(2)}\right)$, with $\mathbf{W}^{(2)} \in \mathbb{R}^{K\times M}, \mathbf{b}^{(2)} \in \mathbb{R}^{K}$. In this matrix-vector multiplication notation the activation functions $h^{(i)}(\cdot)$ are meant to be applied element wise.

where $k = 1, \ldots, K$ and K is the number of output values of layer (2). After applying another activation function to calculate the activations of layer (2), the steps in Eq. (3.12) and (3.13) can be repeated with varying output dimensions. The final target output is chosen to have the desired dimension of the target variable and uses an activation function based on the nature of the data and the assumed distribution of the target variable [3]. Fig. 3.3 shows such a network with an input layer, a hidden layer and the output layer.

Eq. (3.12) and (3.13) make clear that the final output values are dependent on the parameter matrices $\mathbf{W}$ and the parameter vectors $\mathbf{b}$ of each layer. Note that from here on, the bias term $\mathbf{b}$ is omitted and included as a column vector $\mathbf{w}_0 = b$ in $\mathbf{W}$ and a constant column $\mathbf{x}_0 = (1, \ldots, 1)$ in the input $\mathbf{X}$ of each layer as shown in Fig. 3.3. Thus the training of these networks with N layers is about finding optimal values for all $\mathbf{W}^{(i)}$ for $i = 1, \ldots, N$, given a *loss* $L = E(\mathbf{y}, \hat{\mathbf{y}}) + R(\mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(N)})$ which is to be minimized. The first term is called an *error function* and evaluates the network output $\hat{\mathbf{y}}$ against the ground truth $\mathbf{y}$, whereas the second term serves as a regularization of the network weights to prevent the network from overfitting. The minimization step can be solved with *gradient descent* approaches where small changes to the parameters are backpropagated through the network after a single datapoint or a whole batch has been processed and their target values have been predicted, as described in Bishop [3].

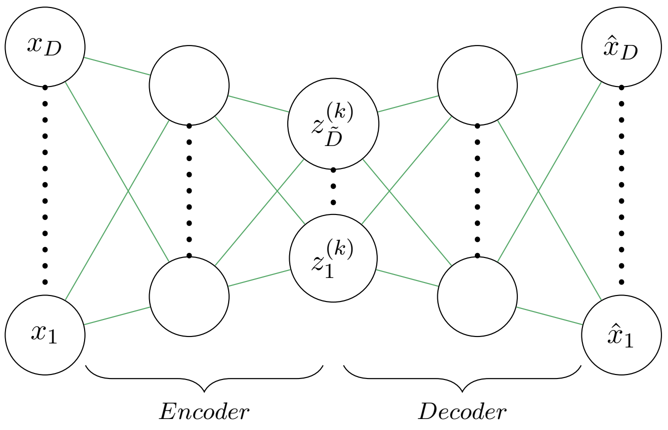


Fig. 3.4: An autoencoder taking inputs $\mathbf{x} \in \mathbb{R}^D$. Encoder compresses the input into a $\tilde{D}$ dimensional deep feature space. The encoded representation at layer *(k)* $\mathbf{z}^{(k)} = (z_1^{(k)}, \dots, z_{\tilde{D}}^{(k)})$ then serves as input for the decoder, whose output $\hat{\mathbf{x}}$ aims to reconstruct the original input $\mathbf{x}$.

3.2.2 Autoencoder

Autoencoder refer to a specific architecture of neural networks and fall into the category of *unsupervised learning*, i.e. the type of learning for which no target variable exists. They aim to encode the input into a lower dimensional deep feature space and reconstruct the original input based on the encoding. The architecture of the network consist of an *encoder* (Fig. 3.4 left network branch), which is the part from the input to the encoding layer and a *decoder* (Fig. 3.4 right network branch) that is a layer-wise mirroring of the encoding, thus the decoder has the input size of the encoder's output dimension and an output size of the encoder's input dimension. The *error function* $E(\mathbf{x}, \hat{\mathbf{x}})$ moreover evaluates on the input data $\mathbf{x}$ instead of a target variable as described in 3.2.1. An example autoencoder is given in Fig. 3.4, where the input $\mathbf{x}$ is of dimension D, and the *encoded input* has dimension $\tilde{D}$. The task of training an autoencoder can therefore be described as finding a compact representation of the input data, that is capable of all information the original data contains. Autoencoder further achieve to reconstruct the original data from the encoded representation. Related work [33, 34] has shown that the transformed input builds a robust and rich set of features. It is possible to use this property as a *feature mapping*, such that $y_i = f\left(g\left(\mathbf{x}_i\right)\right)$, with the encoding function $g : \mathbb{R}^D \mapsto \mathbb{R}^{\tilde{D}}$. In Fig. 3.4 this would mean model f uses the activations $g(\mathbf{x}_i) = \mathbf{z}_i = (z_{i1}, \dots, z_{i\tilde{D}})$ as input instead of the original features $\mathbf{x}_i = (x_{i1}, \dots, x_{iD})$. This procedure has been applied to the relations of compounds and scenarios as elaborated in Section 3.2.3.

An extension to autoencoder is the so-called *denoising autoencoder (DAE)*. As the name implies, the input is augmented by adding noise. This noise can either be a vector added to the input, sampled from $\mathcal{N}(\mu, \sigma^2)$ with μ and σ as the mean and standard deviation of the input data, or by randomly setting weights of the first layer

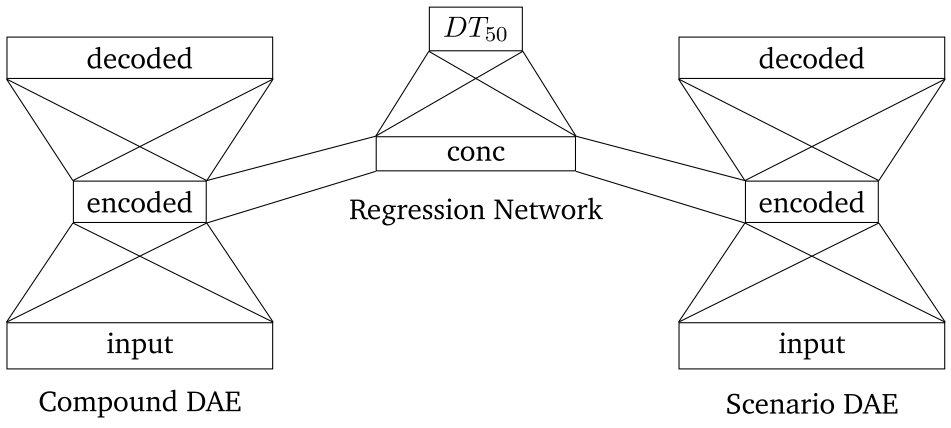


Fig. 3.5: The architecture of the network consists of two separate denoising autoencoder, one for compounds and one for scenarios. The encoded layers are concatenated and form the input for a third network which sits on top of the DAEs and uses the encoded features to predict the DT_{50} value.

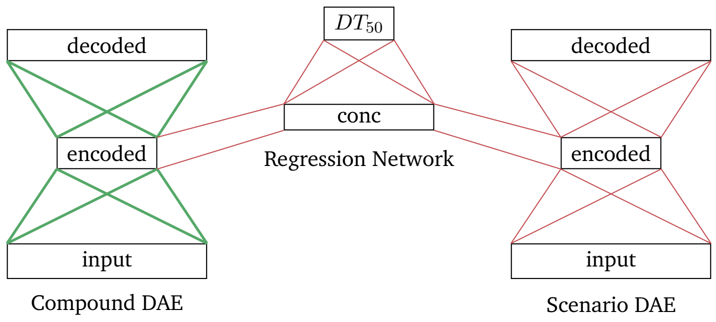
$\mathbf{W}^{(1)}$ to zero and thus blocking some input features. The rest of the autoencoder's architecture is kept the same, except that the error function $E(\mathbf{x}, \hat{\mathbf{x}})$ is still being evaluated on the original input instead of the noisy data. Denoising autoencoder aim to approach a *good representation*, which can be described as *one that can be obtained robustly from a corrupted input and that will be useful for recovering the corresponding clean input* according to Vincent et al. [34]. To achieve such a *good representation*, the input data has to fulfill the constraints of having a robust high level representation.

3.2.3 Denoising Autoencoder Regression Network

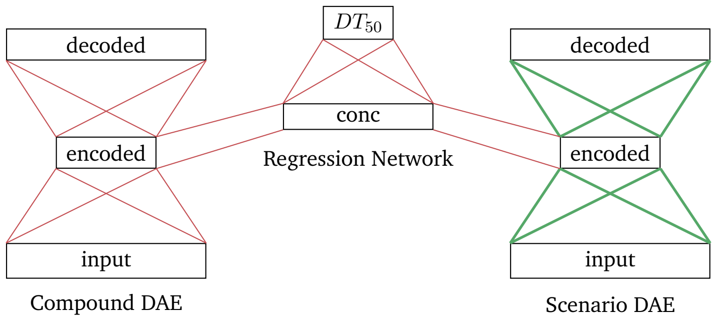
The architecture of the neural network, that is used for the half-life regression, is outlined in Fig. 3.5. It consists of two denoising autoencoder, one for compounds and one for scenarios. The encoding layers of both DAEs are further concatenated and serve as input for the actual regression network part, with the DT_{50} value as target variable.

Training Phases

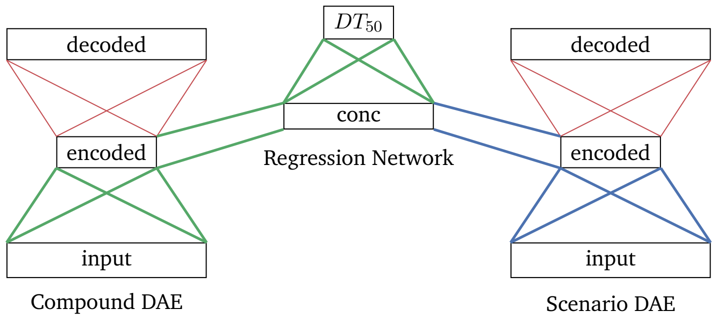
The training of the network can be divided into four phases (see Fig. 3.6 (a) to (d)). Algorithm 1 outlines the pseudo code for the below described procedure. The colors of the paths between layers explicitly state which weights are trainable and used (green), frozen but used (blue) and frozen and unused (red) during the specific phases. A frozen weight does not receive any updates. The training set is further split into 90% training data and 10% validation data against which the corresponding network parts are validated after each epoch. The first two phases serve as initialization phases for the denoising autoencoder. In the first phase (a), the



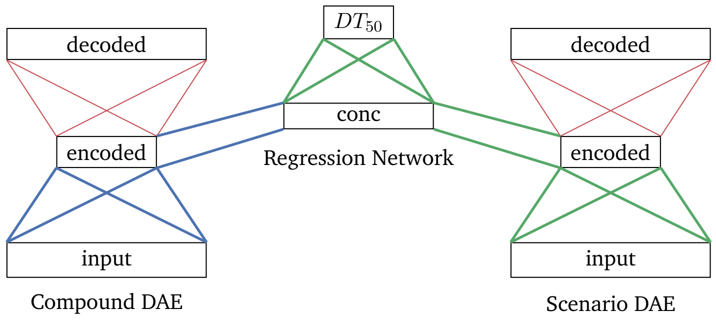
(a)



(b)



(c)



(d)

Fig. 3.6: Training approach for the DT_{50} regression network. (a) and (b) show the initialization phase for compound and scenario DAE respectively. (c) and (d) are the iterative training cycles, which disable the encoding weights of the compound DAE and scenario DAE successively. Red paths indicate frozen weights (disabled weight updates), whereas green paths are trainable weights. Blue paths in (c) and (d) hint to the use of layer activations by giving corresponding input but ignoring weight updates on backpropagation.

compound DAE is trained separately on compounds of the training set , while in the second phase (b) the scenario DAE is trained separately on scenarios of the training set. In both phases all weights except for those of the respective autoencoder are disabled and unused. Starting with phase three the network now aims to optimize for the regression of the DT_{50} value. Therefore, in (c) the decoder of both DAEs are disabled. Additionally, the regression network uses the encoder of the DAEs and successively allows their weights to be updated, such that the encoded representation of compounds and scenarios is further optimized towards the objective to predict precise DT_{50} values. Hence, phase three (c) uses instances of the form $\mathbf{x}_i = (\mathbf{c}_i, \mathbf{s}_i)$ as input, where $\mathbf{c}_i$ serves for the compound encoder and $\mathbf{s}_i$ for the scenario encoder.

The two encoded inputs are then concatenated in the *conc* layer of Fig. 3.5 and used as inputs for the middle network. This means that phase three still uses the scenario encoder as feature mapping for the scenarios but does not update its weights (blue paths in Fig.3.6c) while performing backpropagation. In phase four (d) the compound encoder weights are disabled but used as feature mapping and the scenario encoder weights are trainable again. The network repeatedly executes phase three and four. This learning approach aims to iteratively optimize the encoder weights step by step without making too big steps by prohibiting both network parts to update at the same time, which could lead to a more secure way to find a minimum for this specific setup. The switches between enabling weights of the compound encoder and scenario encoder introduce another parameter, namely the *switch rate* which controls the scheduling behavior of the network. A switch rate of *k* corresponds to the network training *k* epochs before switching between phase three and four. Therefore, the network accomplishes a total of $n_{switches} = \frac{n_{epochs}}{k}$ switches.

Algorithm 1: Regression network training approach.

```
$dae_C, dae_S$ ←initDAEs();
$regNet$ ←initRegressionNetwork($dae_C, dae_S$);
train($dae_C$, $compounds$, $epochs$);
train($dae_S$, $scenarios$, $epochs$);
$epochsPerSwitch \leftarrow \frac{numSwitches}{epochs}$;
for $i$ in $[1, \ldots, numSwitches]$ do
    train($regNet$, $DT_{50}$ , $epochsPerSwitch$);
    if $(i \mod 2) == 0$ then
        freeze($dae_C$);
        unfreeze($dae_S$);
    else
        freeze($dae_S$);
        unfreeze($dae_C$);
    end
end
```

Experiments 4

This chapter introduces the setup of the models, validation methods and metrics used to produce the results in Chapter 5. The experiments were implemented in *Python 3.5*, using the libraries *scikit-learn* [26] for baseline machine learning algorithms, validation methods and metrics, *Keras*[1] for the construction of neural networks, *NumPy* [32] and *SciPy* [2] for further computational purposes.

4.1 Model Validation

To validate a model's performance it is common to split the dataset into two exclusive subsets, the *train* set and the *test* set. As their names imply, the training set is used to build the model. The model thus learns the patterns in the data based on what it explores in the training set. Its performance can then be estimated by predicting instances of the test set. Since the ground truth of the test set is known, it is possible to compare the model's output $\hat{\mathbf{y}}$ to the true values of the target variable $\mathbf{y}$ and apply an arbitrary metric on both vectors. This procedure is called *holdout* and ensures to prevent the model from producing biased predictions since the data points in the test set are unseen and thus help to estimate the generalization performance. A common split of the dataset is to use $2/3$ of the instances as training examples and $1/3$ as test examples. Holdout introduces two problems: 1. Not all instances of the dataset are used for training and testing, 2. if the dataset is too small, the model's performance can vary a lot since the training set may does not represent the data's source distribution anymore.

An approach which addresses the issues from holdout is the so-called *k-fold cross validation*, in which the dataset is split into k exclusive *folds* or subsets. For each subset $\mathbf{X}_i \in \{\mathbf{X}_1, \dots, \mathbf{X}_k\}$ the model is once trained on the training set $\mathbf{X}_{train} = \cup_{j \neq i} \mathbf{X}_j$ and tested against the test set $\mathbf{X}_{test} = \mathbf{X}_i$ which was held out during training. This ensures that all data points of the dataset are used to estimate the generalization performance.

Using two different subsets of the original data for training and testing further requires modifications to some preprocessing steps. That is, the parameters for standardizing the input features as described in Chapter 2 now need to be calculated separately on the training set with the mean $\boldsymbol{\mu}_{train} = E\left[\mathbf{X}_{train}\right]$ and the standard deviation $\boldsymbol{\sigma}_{train} = \sqrt{E\left[\mathbf{X}^2_{train}\right] - \left(E\left[\mathbf{X}_{train}\right]\right)^2}$. The test set is afterwards scaled by

[1]François Chollet et al. *Keras*. [accessed 18.06.2017]. `https://github.com/fchollet/keras`. 2015–

[2]Eric Jones, Travis Oliphant, Pearu Peterson, et al. *SciPy: Open source scientific tools for Python*. [accessed 18.06.2017]. `http://www.scipy.org/`. 2001–

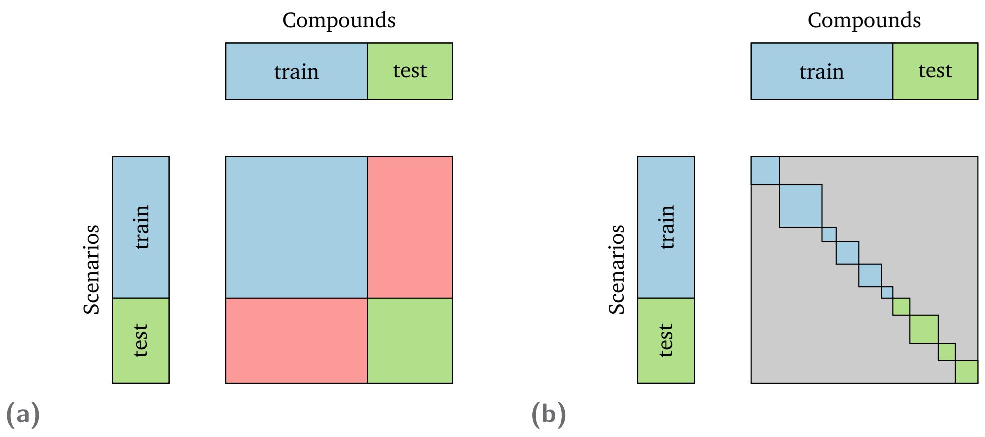


Fig. 4.1: Generating train/test splits on a dataset consisting of two relations. (a) shows a common strategy which removes datapoints (red blocks) having compounds from the train and scenarios from the test set or vice versa. (b) is a improvement over the approach of (a) where the sparseness of the data is used to find groups of datapoints that are distinct without the necessity of removing examples. Those groups are indicated by blocks on the diagonal. The groups are then assigned either to the training set (blue blocks) or to the test set (green blocks). The gray zone represents missing values.

$\mathbf{X}'_{test} = \frac{\mathbf{X}_{test} - \boldsymbol{\mu}_{train}}{\boldsymbol{\sigma}_{train}}$. The same applies for parameterized imputation methods, such that the parameters need to be learned on the training set and the imputation is done on the test set with the training parameters. This separation further prevents to include information from the training set into the test set which would affect the model and produce biased predictions towards the test set.

As the data occurs in two relations (see Chapter 2) it cannot be split trivially into k folds. Each datapoint $\mathbf{x}_i$ consists of a tuple $(\mathbf{c}_i, \mathbf{s}_i)$, namely the compound and the scenario, which corresponds to one half-life y_i. The dataset further has instances $\mathbf{x}_i$ and $\mathbf{x}_j$ where $\mathbf{c}_i = \mathbf{c}_j$ and $\mathbf{s}_i \neq \mathbf{s}_j$. Since $\mathbf{x}_i \neq \mathbf{x}_j$ it looks reasonable to put $\mathbf{x}_i$ into the training and $\mathbf{x}_j$ into the testing set. Nevertheless, this must be avoided as a model could, for example, easily overfit onto the features of examples from the compound relation $\mathbf{X}_c$. Therefore, it is necessary to split the data groups by the following rule: two instances $\mathbf{x}_i$ and $\mathbf{x}_j$ are in the same set, if they either share the same compound $\mathbf{c}_i = \mathbf{c}_j$ or the same scenario $\mathbf{s}_i = \mathbf{s}_j$. Usually this leads to the necessity of not using some examples in the data set at all. This is the case when two datapoints are exclusive, i.e. they do not share a compound or scenario $\mathbf{c}_i \neq \mathbf{c}_j$ and $\mathbf{s}_i \neq \mathbf{s}_j$ but a third instance $\mathbf{x}_k$ exists with $\mathbf{c}_k = \mathbf{c}_i$ and $\mathbf{s}_k = \mathbf{s}_j$. This separation is shown in Fig. 4.1a. Red blocks indicate the described datapoints which need to be left out while splitting.

The sparseness of the target DT_{50} matrix allows the use of a more advanced splitting approach. The target matrix can be seen as a graph where each node represents a tuple. Two nodes are connected if two tuples either share the same compound or the same scenario. The goal is to remove nodes such that the graph is disconnected into two subgraphs, namely the train and the test set. Since the given data is sparse, many nodes are missing anyway and thus the nature of the graph representing the DT_{50} matrix is already disconnected. Each subgraph is equivalent to a group of instances which are not allowed to be separated. The given half-life data has 408 of such groups, which makes it a good fit for this advanced splitting method. Therefore, it is only necessary to simply choose certain groups to be used in the training set and all other groups in the test set, as shown in Fig. 4.1b. By that it is possible to circumvent to remove single instances and all data can be used.

4.2 Metrics

Results in Chapter 5 were evaluated with a relative metric that is based on the *mean squared error*

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n} \sum_{i}^{n} (y_i - \hat{y}_i)^2 \quad , \tag{4.1}$$

which calculates the squared difference between an instance's true DT_{50} value y_i and the model's prediction $\hat{y}_i$. For a batch of instances the *MSE* is the sum over the single squared differences divided by the number of instances n. Even though the *MSE* has a lower bound of 0, which is the result of a perfect prediction $\mathbf{y} = \hat{\mathbf{y}}$, it is unbound in the positive range. This means the interpretation of a *MSE* value is always subjective in the means of how good or bad it actually is. Another problem with the mean squared error is the comparability with model results for different target variables, since its dimension is the squared target variable's dimension.

Therefore, a relative error function was used as metric, which is the *coefficient of determination*

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum_i^n (y_i - \hat{y}_i)^2}{\sum_i^n (y_i - \overline{y})^2} \quad , \tag{4.2}$$

with $\overline{y}$ as the mean value of the ground truth $\mathbf{y}$. The R^2 score is zero if each $\hat{y}_i$ is equal to $\overline{y}$, that is the equivalent to a model that always predicts the target value's mean. A R^2 score of one implies a perfect prediction where $\hat{\mathbf{y}} = \mathbf{y}$. Therefore, a R^2 score between zero and one indicates that the model has explored patterns in the input variables that relate to the target variable. The R^2 score further provides a way to compare models which were trained on different parts of the dataset, since its value is independent of the target variable's dimension.

4.3 Model Setups

This chapter gives a brief overview of the setup from which the results in Chapter 5 were produced. For the baseline models (see Section 4.3.1), this refers to a description of fine-tuning parameters in Tab. 4.1. In Section 4.3.2 the implementation and the available parameters of the neural network are described.

4.3.1 Baseline Models

The algorithm implementations for baseline models were taken from the *scikit-learn* library which provides a broad support on curated machine learning methods. Parameters that are described in the following were determined by evaluating a *grid search* on a separate validation set. This optimization step is done by creating a parameter grid consisting of all the combinations of possible parameters given. In the case of two parameters with 20 values each, this would result in a parameter grid of 400 combinations. A 10-fold cross validation is then performed for each parameter combination. The optimal parameters are finally chosen by the model yielding the best results.

Random Forest

The implementation of *Random Forest* makes use of decision trees based on an enhanced version of the *Classification and Regression Tree (CART)* algorithm [5]. The model was tested with random feature subspaces of different size, beginning with 5 features and successively adding more until all have been used. To further prevent overfitting it is possible to choose two more parameters. For one the *max_depth* regulates the tree growth by only allowing trees of depth *max_depth*. The second parameter is *min_samples_split* that allows to set a lower limit on the number of samples required to split at a node. If this number is undershot, the node will be a leaf. The parameters *max_features*, *max_depth* and *min_sample_split* were obtained by evaluating a grid search over the ranges listed in Tab. 4.1.

Support Vector Regression

Support Vector Regression (SVR) is a slight modification to the method presented in 3.1.3 which allows regression instead of classification. The implementation in *scikit-learn* is based on the *libsvm* library [6]. Results shown in Chapter 5 compare different kernels, such as `linear`, `poly` and `rbf`. The `poly` kernel further gives another degree of freedom, namely the polynomial degree d in Eq. (3.10). For the `rbf` kernel the same applies to the γ parameter where $\gamma = \frac{1}{2\sigma^2}$ in Eq. (3.11). Besides these two kernel specific parameters, the SVR algorithm allows a trade off between the model errors on the training data and margin maximization with the C parameter. Furthermore, the ϵ parameter specifies the ϵ-tube within which no penalty is assigned to training samples with an absolute prediction error of less than ϵ. The parameter combinations are shown in Tab. 4.1.

Model	Parameter	Min.	Max.	Step Size	Grid Size
Random Forest	max_depth	5	D	5	$\frac{D}{5} \times \frac{D}{5} \times 100$
	$max\_features$	5	D	5	
	min_sample_split	2	200	2	
SVR linear	C	2^{-3}	2^{15}	$\times 2$	100
	ϵ	0.1	1.0	0.1	
SVR poly	C	2^{-3}	2^{15}	$\times 2$	400
	ϵ	0.1	1.0	0.1	
	$degree$	2	5	1	
SVR rbf	C	2^{-3}	2^{15}	$\times 2$	1000
	ϵ	0.1	1.0	0.1	
	γ	2^{-15}	2^{3}	$\times 2$	

Tab. 4.1: Parameter ranges for Random Forest and Support Vector Regression with linear, polynomial and rbf kernel. Each parameter is given a range through the minimum and maximum value and additionally a step size at which the parameter space is explored. An exhaustive grid-search on a validation set over all parameter combinations was performed for each model to obtain the optimal values given the training data. The maximum $max\_features$ and max_depth value D refer to the number of features in the input space.

4.3.2 Denoising Autoencoder Regression Network

For quick prototyping of the network, the library *Keras* was used, which is based on *TensorFlow* [1]. All layers are fully connected and use the *LeakyRelu* activation function which is $h(x) = x$ for $x \geq 0$ and $h(x) = 0.01x$ for $x < 0$ to break linearity and avoid dead neurons as described by Maas et al. [19]. After each layer, except for the output layers, *batch normalization* [13] is applied, which allows the network to normalize the activations on the batch axis if necessary. To ensure faster convergence, *xavier initialization*, as proposed by Glorot and Bengio [11], is used as weight initialization method before the training starts. Regularization terms are applied to the weights and activations in the form of the *L2* norm

$$R_{weights} = \sum_i ||\mathbf{W}^{(i)}||_2^2 \tag{4.3}$$

$$R_{acts} = \sum_i ||\mathbf{z}^{(i)}||_2^2 \tag{4.4}$$

and incorporated into the loss function defined in Eq. (4.5). The compound DAE uses the sigmoid function $h(x) = \frac{1}{1+\exp(-x)}$ as output activation, since the target is a bit-vector, whereas the scenario DAE as well as the regression network both use a linear output activation due to the continuous nature of their target variables. To prevent the networks from overfitting, an *early stopping* approach is used, stopping the training phase as soon as the network's performance on a separate validation set does not improve in a streak of five *epochs* (one epoch corresponds to feeding the complete dataset through the network once). All three network parts are trained individually to minimize the objective function

$$L = MSE(\mathbf{y}, \hat{\mathbf{y}}) + \lambda_1 R_{weights} + \lambda_2 R_{acts} \tag{4.5}$$

with the *mean squared error* as of Eq. (4.1), regularization factors $\lambda_1 = \lambda_2 = 0.01$ and their input, output, weights and activations of each part respectively. The optimization is done by the *Adam optimizer* [14].

The network architecture described in Section 3.2.3 exposes several parameters whose influence on the model's performance is investigated in Chapter 5, that is

- *Batch size*: Number of input samples gathered in mini-batches [18] to feed through the network before updating the network weights. Low values potentially create batches that do not represent the distribution of the whole dataset well enough and thus generate noisy gradients in the optimization step. High values effectively lead to a slower convergence time, since each epoch generates a lower number of weight updates.
- *Noise factor*: Scale η of the noise which is added to each input instance of the compound and scenario DAE by $\mathbf{x}_{noisy} = \mathbf{x} + \eta\mathbf{v}$, $\mathbf{v} = (v_1, \ldots, v_m)$ and $v_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ where m is the number of features in $\mathbf{x}$ and μ_i and σ_i is the mean and standard deviation of the *i*-th column of $\mathbf{X}$.
- *Compression factor*: Dimensionality reduction factor *cf* between two successive layers. If layer *i* has *n* outputs, then the layer $i+1$ will be of size $\texttt{ceil}(cf \cdot n)$. The compression factor therefore directly influences the depth of the networks, since layers are added as long as the term $\texttt{ceil}(cf \cdot n)$ is not smaller than the encoding dimension for the respective DAE or the output dimension of the regression network. The value is bounded by $0 < cf < 1$. A higher compression factor leads to a lower number of layers and thus less weights in the network, whereas a lower rate stacks more layers and consequently introduces more weights.
- *Encoding dimension*: Dimension of the encoding layer for compounds and scenarios. A lower dimension translates in a more dense representation of the input, which influences the quality of the encoded features. The encoding dimension should be lower than the original input dimension to achieve a compression at all. It furthermore controls the depth of the network since a lower encoding dimension requires more layers to compress the input dimension with a fixed compression factor.

Furthermore, each network part is trained 100 epochs. The *learning rate*, that is the step size of the weight updates during optimizing the loss function in Eq. (4.5), was chosen to be 0.01, as the Adam optimizer uses *dampen* and *decay* to adjust the rate accordingly (see Kingma and Ba [14]). The evaluation was performed with 10-fold cross validation as described in Section 4.1.

4.4 Permutation based P-Value Tests

To make sure that a model found a signal in the data and the scores of Section 4.2 are not generated by chance, it is possible to perform *p-value based permutation tests* to study a model's performance, as proposed by Ojala and Garriga [25]. The p-value in this context is defined as

$$p = \frac{\left|\left\{D' \in \hat{D} : e\left(f, D'\right) \leq e\left(f, D\right)\right\}\right| + 1}{k+1} \tag{4.6}$$

where $\hat{D}$ is a set of randomized versions of the original dataset D, e is an error function, f is the model to be tested and k is the number of permutations $\left|\hat{D}\right|$. It represents the fraction of samples from $\hat{D}$ for which the model f had a lower error $e(f, D')$ than in the original dataset D. This can be described as the likelihood of the model achieving the error $e(f, D)$ by chance. Ojala and Garriga [25] therefore propose the following two permutation tests to study a model's performance. Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ be the original dataset:

- **Test 1 (Permute labels)** Let π be a permutation of n elements. The randomization process is generating $D' = \{(\mathbf{x}_i, \pi(y_i))\}_{i=1}^{n}$, i.e. shuffling the target vector $\mathbf{y}$ with the permutation π.
- **Test 2 (Permute data columns per class)** Let $\mathbf{X}(c) = \{\mathbf{X}_i | y_i = c\}$ of size $l_c \times m$ be the set of rows of $\mathbf{X}$ with target class c and $\pi_1, \ldots, \pi_m$ be m independent permutations of l_c elements. Then $\mathbf{X}(c)' = \left[\pi_1\left(\mathbf{X}(c)^1\right), \ldots, \pi_m\left(\mathbf{X}(c)^m\right)\right]$ is a randomized version of $\mathbf{X}(c)$ in which each π_j is applied independently to the column $\mathbf{X}(c)^j$. Repeating this for all values c of the set of possible classes Y generates $\mathbf{X}' = \left\{\mathbf{X}\left(c\right)' | c \in Y\right\}$ and finally $D' = \{(\mathbf{x}'_i, y_i)\}_{i=1}^{n}$.

The final step for both tests is the calculation of the p-value as of Eq. (4.6). Test 1 aims to examine whether the model found a connection between the data $\mathbf{X}$ and its class labels $\mathbf{y}$, whereas Test 2 checks out if there exists a significant dependency between features of $\mathbf{X}$ that may reduce the error score. Ojala and Garriga [25] suggest a threshold of $\alpha = 0.05$ to decide whether the result is significant or not.

Since the given target variable DT_{50} is not categorical but numerical, a discretization method was applied to achieve the permutations in Test 2. Therefore, each y has been marked with an interval label, which indicates that the value lies in a specific ten-day interval and serves as the class attribute in the test. Half-lives above 320 days have been discretized into one single class.

Results 5

The previous chapters have introduced the dataset, methods of machine learning and the experimental setup, subsequently this chapter will present and discuss the produced results. First, different preprocessing configurations are evaluated against the baseline models consisting of Random Forest, Support Vector Regression and Bagged SVR, which simply applies the bagging approach from Section 3.1.2 to SVR instead of Decision Trees. Furthermore, the baseline models regression results are visualized and analyzed. In Section 5.3, the denoising autoencoder regression network is extensively examined regarding encoding performance and parameter influence. Finally, the *p*-value tests are evaluated.

5.1 Preprocessing configurations

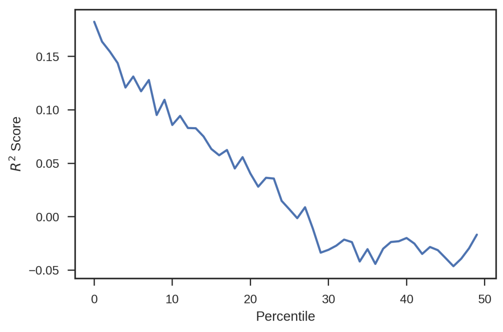
The preprocessing steps listed in Chapter 2 and 4 showed a vast number of fine-tuning steps to achieve a representation that might better suit the exploration methods of machine learning models. Therefore, this section concentrates on the influence of processes that are applied before the model is trained.

Section 2.1 introduced Fingerprinter as transformation functions from a compound's SMILES-String to an informative bit-vector with results of certain queries against the compounds chemical structure. The Fingerprinter tested were MACCS, FP2, FP3 and FP3, with each containing a different set of queries. Tab. 5.1 shows, that the MACCS Fingerprinter achieved the best representation on all baseline models. Even though FP2 and FP4 Fingerprinter generate descriptors of higher dimension than MACCS, they seem not to incorporate more structural information that correlate with the DT_{50} . FP3 queries lack in quality, quantity or both for the models to find a signal that relates to the target variable, according to their R^2 scores around and below zero.

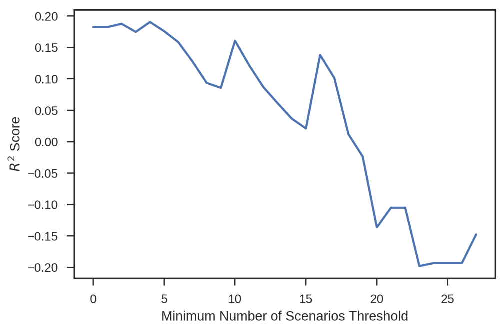
Another step of the preprocessing pipeline is to exclude datapoints that may do not fit into the true distribution, such as outliers, or datapoints that did not

Model	MACCS	FP2	FP3	FP4
Random Forest	0.182	0.181	0.001	0.143
SVR rbf	0.176	0.052	-0.177	0.096
Bagged SVR rbf	**0.185**	0.070	-0.086	0.126

Tab. 5.1: R^2 scores for models evaluated against datasets with compound descriptors generated from MACCS, FP2, FP3 and FP4 Fingerprinter.



(a)



(b)

Fig. 5.1: R^2 scores for different approaches of successive data exclusion. (a) drops the lowest and highest p% DT_{50} values, whereas (b) removes those compounds that do not have a certain number of scenarios.

give enough information about the relation to their DT_{50} value. This is further evaluated in Fig. 5.1a which compares a Random Forest's R^2 score against the successive removing of percentiles at the half-life borders, that is, the p% of the datapoints with the lowest and highest half-lives. The strip off was based on the idea that outstanding short and long DT_{50} values were only approximately measured in dossier studies and may stem from other degradation processes. However Fig. 5.1a clearly shows a constant loss of signal when dropping the border percentiles. Furthermore, 5.1b shows a comparison of dataset evaluations of those that only contain compound-scenario combinations where the compound exists with at least a given threshold as number of scenarios. This tests, whether the intra-compound DT_{50} relationship is better explored if each compound is given enough information (threshold represented by number of scenarios). The results suggest, that dropping compounds with less than five scenarios hardly effects the signal detection. With an increasing scenario threshold, there is a reasonable loss of information.

At the end of Section 2.2, the problem of missing values in the scenarios relation is introduced. This can be addressed by different imputation methods. For the results in Tab. 5.2, each method generated a separate dataset which was evaluated against the models. The results reveal that the advanced methods such as *MICE*, *SoftImpute* and *Matrix Factorization* do not provide a better imputation, regarding the R^2 scores, than the basic approach of mean imputation for all models. This either suggests, that the imputation works equally bad with the given data, or the imputed values do not fit to the dependencies between the compounds and their associated half-lives.

Model	Mean	KNN	MICE	SoftImpute	Matrix Fact.
Random Forest	0.182	0.174	0.168	0.168	0.160
SVR rbf	0.176	0.174	0.178	0.166	0.166
Bagged SVR rbf	**0.185**	0.183	**0.185**	0.173	0.173

Tab. 5.2: R_2 score results for different imputation methods of missing values in scenarios.

Model	Parameter	Best Value	R^2 Score
Random Forest	max_depth	14	0.182
	$max\_features$	14	
	min_sample_split	22	
SVR linear	C	2^{-3}	-0.043
	ϵ	1.0	
SVR poly	C	2	-70.9
	ϵ	0.1	
	$degree$	3	
SVR rbf	C	2	0.176
	ϵ	0.3	
	γ	2^{-7}	

Tab. 5.3: Local optimal parameter values with their validation result for Random Forest and Support Vector Regression with the linear, polynomial and rbf kernel. The values have been determined by an exhaustive grid-search as described in Section 4.3 with a parameter-space as of Tab. 4.1.

5.2 Baseline Models

The baseline models introduced in Section 3.1 expose several parameters that were optimized on a separate parameter-optimization test set. The optimal parameter values are listed in Tab. 5.3, with their respective results on the test set. It should be mentioned that those values are no global optimums since the grid-search only exposes results from a discrete grid. Hence there might be an undetected parameter combinations that lie between two grid points and lead to better results. The R^2 score for the linear SVR indicates that there is no simple linear dependency between the features and the target variable. Additionally, the correlation can not be expressed as polynomial function as the R^2 score of -70.9 for the SVR with polynomial kernel suggests. The only two models that are capable of exploring dependencies better than a simple average predictor (R^2 score > 0) are the Random Forest and SVR with radial basis function kernel. Interestingly the best value for Random Forest's *min_features* is approximately the square root of the input dimension D, which is, next to $log_2(D)$, a widely suggested value for this parameter.

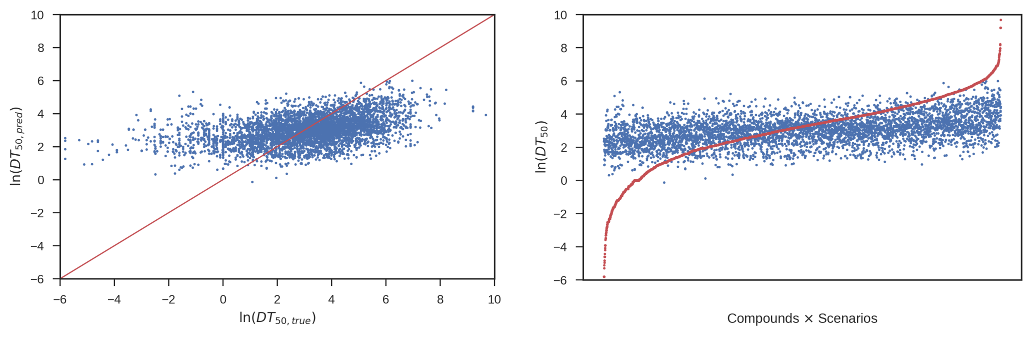


Fig. 5.2: Graphical evaluation of the predicted and true DT_{50} values. Predictions in (a) and (b) were generated with a Random Forest model. (a) compares the predicted and true value for each datapoint (blue dots). The red line would be a perfect prediction where $\hat{\mathbf{y}} = \mathbf{y}$. (b) lists all compound-scenario tuples, sorted by their associated DT_{50} value (red dots). Blue dots correspond to each predicted half-life.

Using the findings from above, Fig. 5.2 visualizes the DT_{50} predictions of a Random Forest model. In (a) the true half-lives (x-axis) are plotted against the predicted values for each datapoint (y-axis) as blue dots. A perfect prediction is marked by the red line, where $\hat{\mathbf{y}} = \mathbf{y}$. The plot shows a slight slope towards the correct direction, though prediction errors equally occur below and above the ground truth. Next to that, (b) shows the predicted DT_{50} values (blue dots) for each datapoint, sorted ascending by its true half-life in red. This shows, that the model seems to predict values around the mean of the training data with high variance. The predictions deviate at an extreme rate from the true values in ranges below $\ln(0.5) \approx 1.65$ and above $\ln(5.5) \approx 245$ days, which further sustains the assumption that the model only learns a mean approximation with some noise.

5.3 Denoising Autoencoder Regression Network

This section concentrates on the performance and results of the network proposed in Section 3.2.3. At first, the denoising autoencoder's encoding quality on compounds and scenarios is measured against a third standard dataset on a visual and a numeric level in Section 5.3.1. Subsequently, Section 5.3.2 compares results for the regression architecture with different parameters that were introduced at the end of Section 3.2.3. Since combining these parameters and evaluating their impact with a gridsearch as in Tab. 5.3 is not feasible time-wise due to the complexity of the model, each parameter has been assigned a default value as of Tab. 5.4. Their impact on the regression performance has been analyzed in Section 5.3.2.

Parameter	Default Value
batch size	32
number of DAE switches	5
compression factor	0.5
noise factor	0.66
compound enc. dim.	32
scenario enc. dim.	4
learning rate	0.01
epochs	100

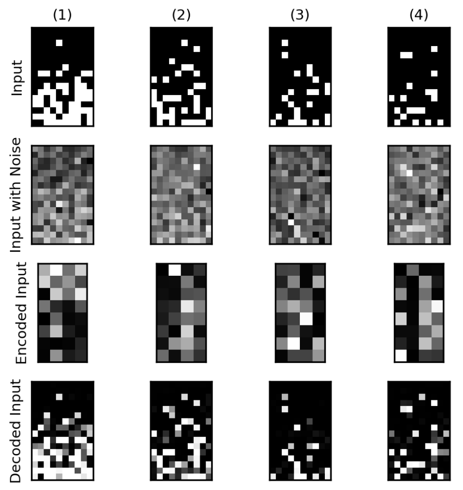
Tab. 5.4: Default values for the network parameters used for generating the results in Section 5.3.

5.3.1 Autoencoder Quality

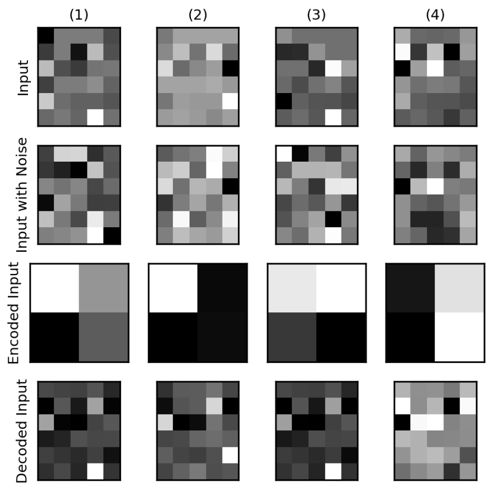
For the network, evaluation it is necessary to begin with the autoencoder performance. To make sure the constructed autoencoder works as desired, it was validated against the MNIST [17] database that consists of 70.000 handwritten digits. Fig. 5.3c illustrates four sampled images from a test subset of this dataset. Each image is drawn in the first row. The second row shows the input image with noise addition of scale $\eta = 1$, followed by the compressed representation of the autoencoder, i.e. the activations of its encoding layer, trained on the MNIST training subset, in row three. The last row consists of the decoded encoding. The results show both, a successful decoding of the compressed image and the elimination of noise. This means the autoencoder has found a *good representation* of the input features in a lower dimensional space which is capable of containing all important features to reconstruct the original image.

To compare these results to the relations from Chapter 2, the same procedure has been applied to the compound dataset in Fig. 5.3a and the scenario dataset in Fig. 5.3b. Each feature of the inputs is expressed as pixel with values between zero (black) and one (white). The results show, that the MACCS descriptors of compounds can be compressed into the lower dimensional encoding space with similar success as of the MNIST images. The DAE also learns to almost completely remove the additional noise suggesting that the autoencoder efficiently extracts robust features out of the high dimensional input space. This is not the case for scenarios as Fig. 5.3b indicates. The decoded scenarios lose important blocks of features as compared to their inputs. Additionally, further noise is present in most parts of the image. These observations also become visible in Fig. 5.4, where the successive loss of each DAE is plotted against the epochs. To make the loss between different datasets comparable the *MSE* term in Eq. (4.5) has been replaced by

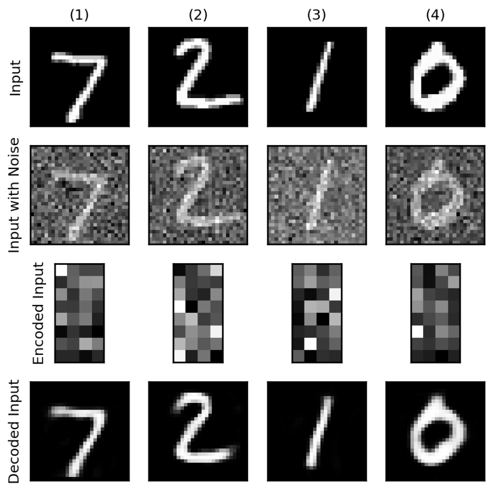
$$\frac{MSE(\mathbf{y}, \hat{\mathbf{y}})}{\frac{1}{n}\sum_i^n (y_i - \overline{y})^2} = 1 - R^2(\mathbf{y}, \hat{\mathbf{y}}) \quad . \tag{5.1}$$



(a)



(b)



(c)

Fig. 5.3: Denoising Autoencoder layer activations. Each figure shows four examples. Each example is represented in its four states: the input, the input with noise addition ($\eta = 1$), the encoded input and the decoded encoding, which is the DAE output. (a), (b) and (c) were generated respectively on the compound, scenario and MNIST datasets.

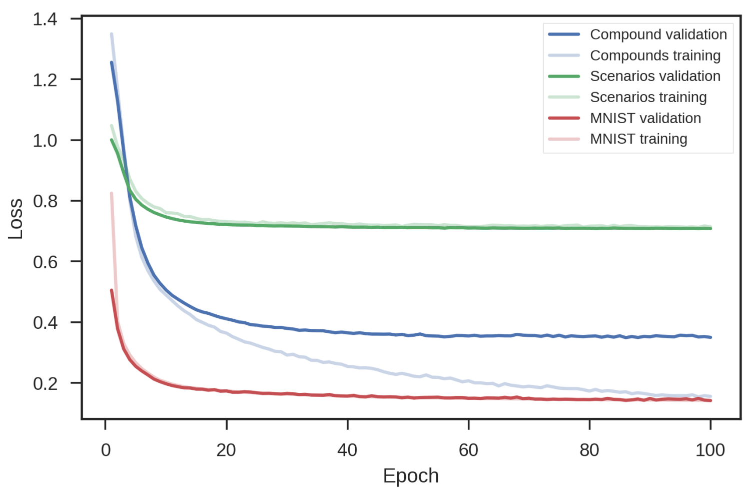


Fig. 5.4: Validation loss of the DAE after each epoch for the compound (blue), scenario (green) and MNIST (red) datasets. The *MSE* in Eq. (4.5) is replaced by its relative score as of Eq. (5.1).

Minimizing the above is equal to maximizing the R^2 score (proof omitted). Fig. 5.4 reveals, that the DAE works slightly worse in minimizing the loss on the compound than the MNIST dataset, whereas the scenario dataset seems to stagnate at a much higher loss which fits with the observations from Fig.5.3. Moreover the loss curve for compounds in Fig. 5.4 points out, that the DAE starts overfitting the training data after epoch 20. The bad performance on scenarios (see green line in Fig.5.4) can have multiple possible sources: First, the environmental condition measurements from the dossiers contain too much noise that makes the input features non-robust. Second, the fraction of missing values (see Tab. 2.2) is too high and the quality of their imputation is too low.

5.3.2 Half-Life Regression

This Section presents results showing the performance of the denoising autoencoder regression network. Fig. 5.5 plots the loss across all epochs of the network using default parameters for training and validation data (see Tab. 5.4). The big margin between the training and validation loss indicates that the network starts overfitting on the training data only after a few epochs. This might be caused by the network weights being underdetermined by the number of training examples. As shown by Zhang et al. [37], neural networks can easily fit a random labeling of the training data as soon as the number of network weights exceed the training sample size. Since this problem cannot be addressed by collecting more data, the following paragraphs describe the search for a network parameter setup that improves the regression R^2 score over those of the baseline models. Section 4.3.2 introduced parameters for the denoising autoencoder regression network, such as the batch

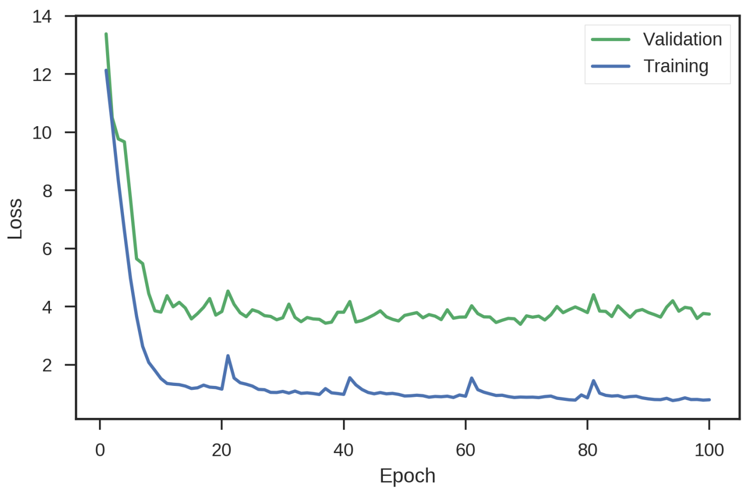
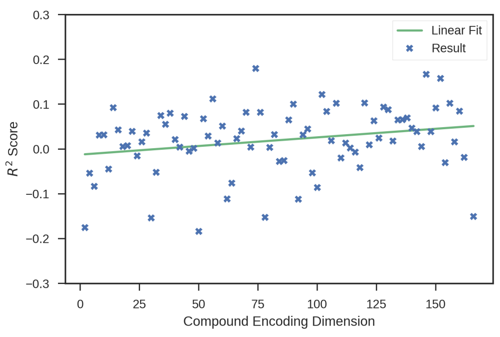
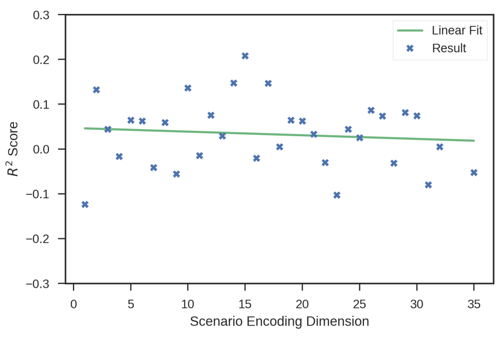


Fig. 5.5: Regression network loss evaluation as of Eq. (4.5) on training data (blue) and validation data (green).



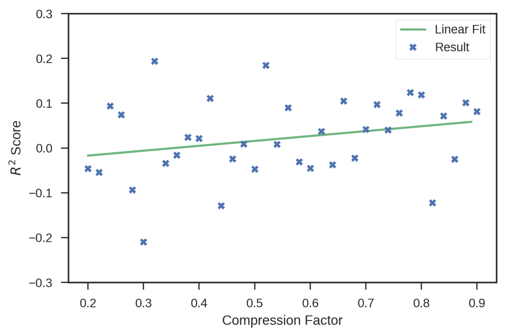
(a)



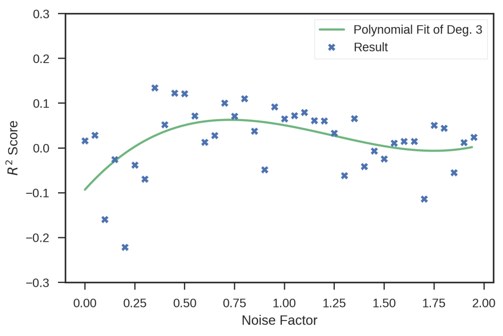
(b)

Fig. 5.6: Regression network R^2 scores for varying encoding dimensions for (a) compounds and (b) scenarios. The encoding dimensions were evaluated between one and the initial input size.

size, noise factor, compression factor and the training phase switch rate that have influence on the regression performance. Fig. 5.6 elaborates the effect of the DAE encoding dimension for both, compounds and scenarios. The intervals are chosen to be $[1, 166]$ for compounds and $[1, 35]$ for scenarios with a step size of two. Even though the R^2 scores in Fig. 5.6a are of high variance, there is a slight slope towards higher scores with increasing encoding dimension. This might indicate that the use of a feature mapping into a lower dimensional space by autoencoder to extract robust features does not or negatively impact the objective of predicting the DT_{50} value for the given data. Fig.5.6b shows no visible effect of the scenario encoding dimension on the regression performance, which might be due to the bad encoding quality of scenarios presented in Fig. 5.3b and Fig. 5.4.



(a)



(b)

Fig. 5.7: Regression network R^2 scores for varying compression factors in (a) and noise factors in (b).

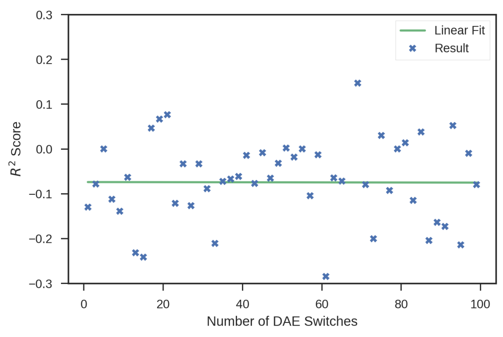
To increase the width and depth of the network we iterated the compression factor from 0.2 to 0.9 in step sizes of 0.02 (see Fig. 5.7a). The lower bound was chosen as the lowest possible compression factor with $166 \times 0.2 \approx 33$ which is approx. the default encoding dimension for compounds and thus constructs an autoencoder that connects its input directly with the encoding layer. The upper bound was set to 0.9 due to memory limitations as the network strongly grows with values near 1.0. The results in Fig. 5.7a are again of high variance and thus hard to interpret, since no clear direction is visible. The noise factor evaluation in Fig. 5.7b seems to take a peak around a value of 0.5 as the fitting curve (green) suggests. This may come from noise with a factor above 0.5 being too much noise for the DAE to remove. Though noise with a factor below 0.5 could be too weak for the desired effect of the DAE building robust features.

Fig. 5.8a shows the R^2 score for values between one and 100 as number of switches between the compound and the scenario DAE as described in Section 3.2.3. The results unravel no clear dependency, which does not necessarily mean there is none but could be hidden behind the high variance in the weak and unstable performance that is present in all parameter evaluations.

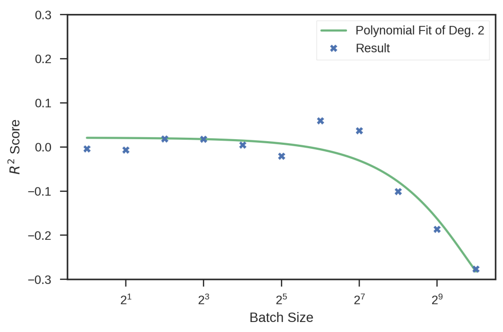
The last parameter tested is the batch size. Fig. 5.8b reveals a slight optimum at a batch size of 64. Lower values could potentially create batches that do not represent the distribution of the whole dataset well enough and thus generate noisy gradients. Higher values seem to effectively worsen the performance, which might come from slower convergence times, as the network was trained on 100 epochs for all batch sizes.

5.4 Permutation based P-Value Tests

The *p*-value tests explained in Section 4.4 were performed with a Random Forest model using its optimized parameters from Tab. 5.3, with 100 permutations. Test



(a)



(b)

Fig. 5.8: Regression network R^2 scores for varying number of switches between the compound and scenario autoencoder in (a) and different batch sizes in (b).

Features	p-value	R^2 mean
all	0.089	0.157
compounds	0.020	0.149
scenarios	0.525	0.171

Tab. 5.5: P-value results for Test 2 from Section 4.4 for permutation of all, only compound and only scenario features.

1 finished with a p-value Eq. (4.6) of 0.01 and an average R^2 score of -0.0427. This indicates, that no permuted dataset achieved a better result than the original and thus that there is a significant dependency between the data $\mathbf{X}$ and its class labels $\mathbf{y}$, which is successfully explored by the model. The second test resulted in a p-value of 0.089 with an average R^2 score of 0.157 on the permuted datasets, that is only slightly worse than the results of 0.182 for the original set. According to the threshold of $\alpha = 0.05$, this means that there is no significant dependency between features of the same class that could be detected by the model. Moreover, this clearly reveals, that the data lacks important information giving explorable structure for the model. Tab. 5.5 further lists a comparison of Test 2 results based on a permutation of all, only compound and only scenario features. The p-value of 0.020 for the compound permutation test indicates a significant dependency between the descriptor features, even though the mean R^2 score of 0.149 shows that there is only a slight margin between the performance on the original and the permuted dataset. This is even worse for the scenario features, with a p-value of 0.525, that seem to randomly perform better on a permuted dataset and thus being of no help for the model to detect any dependency at all. These results evidently clarify, why the baseline models in Section 5.1 and 5.2 achieve only such a poor performance and why the parameter evaluations in Section 5.3 are of high variance and therefore hard to interpret.

Conclusion 6

The main goal of this thesis was to predict the half-lives of environmental pollutants with the help of machine learning. To achieve this, the Eawag-Soil package was introduced and analyzed, providing statistics about molecular compounds, environmental conditions and associated biodegradation half-lives. Chapter 3 presented three main approaches of machine learning: Decision Trees, Support Vector Machines and Neural Networks. For each method an experimental setup has been described in Chapter 4. Moreover, an advanced approach of combining denoising autoencoder for the compound and scenario relations with the DT_{50} regression using scheduled training phases was presented and the influence of its parameters on the regression performance evaluated.

Experiments using baseline models in Section 5.2 revealed a modest dependency between the molecules, the environmental conditions and the associated DT_{50} values in the dataset. The tested models were merely capable of a slightly better regression than a model that always predicts the data's mean half-life. These performances could arise from two sources: 1. The dataset either contains an insufficient number of instances and thus represents only a small set of patterns that are necessary for exploring the full dependency on half-lives, 2. The dataset lacks in informative features that may have been present in the biodegradation process but have not been measured in the dossiers studies. The performance loss while removing data as of Fig. 5.1 backs up the idea of the first possible source, intending that more datapoints should increase the regression performance. The permutation based p-value test results in Section 5.4 strictly support the second hypothesis, showing no expected performance loss when randomizing the scenario features. This shows that the scenarios, as of the shape in the given dataset, currently do not affect the DT_{50} values. The reason for this is the sparseness of the DT_{50} matrix (0.21 % occupation rate), suggesting that having a dataset with more combinations of compounds and scenarios while keeping the number of unique compounds and scenarios the same, thus increasing the occupation rate, should further reveal the influence of the environmental conditions on the half-lives.

Advanced methods of neural networks were not capably of exceeding the baseline models' performances. Their outcomes were rather unstable with high fluctuations, which could be the result of high underdetermination of weights by the relatively small number of training examples as described by Zhang et al. [37].

For future research on this topic the above sources for problems should be resolved. Since capturing more features for existing dossier studies is impossible for the existing dataset, measuring more datapoints would be the next step to do. To improve the autoencoder performances, it may be helpful to train them on separate further datasets. As their objectives are not connected to the half-life regression in the first two training phases of the regression network, external data can be used to focus on the encoding performance.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on p. 23).

[2] D. Aronson, R. S. Boethling, P. H. Howard, and W. Stiteler. „Enhanced Aerobic Biodegradation of Naphthalene in Soil: Kinetic Modelling and Half-Life Study“. In: *Chemosphere* 63 (2006), 1953–1960 (cit. on p. 1).

[3] C. M. Bishop. *Pattern Recognition and Machine Learning*. Ed. by Michael Jordan, Bernhard Schölkopf, and Jon Kleinberg. Information Science and Statistics. Springer, 2006 (cit. on pp. 13, 14).

[4] R. S. Boethling, P. H. Howard, W. Meylan, et al. „Group Contribution Method for Predicting Probability and Rate of Aerobic Biodegradation“. In: *Environmental Science and Technology* 28 (1994), 459–465 (cit. on p. 1).

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Belmont, Calif: Wadsworth International Group, 1984, p. 358 (cit. on pp. 10, 22).

[6] C.-C. Chang and C.-J. Lin. „LIBSVM: A library for support vector machines“. In: *ACM Transactions on Intelligent Systems and Technology* 2 (3 2011). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, 27:1–27:27 (cit. on p. 22).

[7] C. Cortes and V. Vapnik. „Support-Vector Networks“. In: *Machine Learning* 20.3 (1995), pp. 273–297 (cit. on p. 11).

[8] J. L. Durant, B. A. Leland, D. R. Henry, and J. G. Nourse. „Reoptimization of MDL keys for use in drug discovery“. In: *Journal of Chemical Information and Computer Sciences* 42 (2002), pp. 1273–1280 (cit. on p. 4).

[9] EU. „Regulation (EC) No. 1107/2009 of the European Parliament and of the Council of 21 October 2009 concerning the placing of plant protection products on the market and repealing Council Directives 79/117/EEC and 91/414/EEC.“ In: *Official Journal of the European Union 2009, L 309* (), pp. 1 –50 (cit. on p. 7).

[10] K. Fenner, S. Canonica, B. I. Escher, et al. „Developing Methods to Predict Chemical Fate and Effect Endpoints for Use Within REACH“. In: *Chimia* 60 (2006), 683–690 (cit. on p. 1).

[11] X. Glorot and Y. Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS-10)*. Ed. by Yee W. Teh and D. M. Titterington. Vol. 9. 2010, pp. 249–256 (cit. on p. 23).

[12] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction*. 2nd ed. 2009. Corr. 7th printing 2013. Springer Series in Statistics. Springer, Apr. 2011 (cit. on p. 11).

[13] S. Ioffe and Christian S. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015* (2015), pp. 448–456 (cit. on p. 23).

[14] D. P. Kingma and J. Ba. „Adam: A Method for Stochastic Optimization“. In: *Computing Research Repository* abs/1412.6980 (2014) (cit. on pp. 24, 25).

[15] R. Kuhne, R. U. Ebert, and G. Schuurmann. „Estimation of Compartmental Half-lives of Organic Compounds – Structural Similarity versus EPI-Suite“. In: *QSAR and Combinatorial Science* 26 (2007), 542–549 (cit. on p. 1).

[16] D. Latino, J. Wicker, M. Gütlein, et al. „Eawag-Soil in enviPath: a new resource for exploring regulatory pesticide soil biodegradation pathways and half-life data“. In: *Environmental Science: Process & Impact* (Feb. 23, 2017). published (cit. on pp. 1, 3, 6, 7).

[17] Y. LeCun and C. Cortes. *MNIST handwritten digit database*. 2010 (cit. on p. 31).

[18] M. Li, T. Zhang, Y. Chen, and A. J. Smola. „Efficient Mini-batch Training for Stochastic Optimization“. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: ACM, 2014, pp. 661–670 (cit. on p. 24).

[19] Andrew L. M., A. Y. Hannun, and A. Y. Ng. „Rectifier nonlinearities improve neural network acoustic models“. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 23).

[20] K. Mansouri, T. Ringsted, D. Ballabio, R. Todeschini, and V. Consonni. „Quantitative Structure-Activity Relationship Models for Ready Biodegradability of Chemicals“. In: *Journal of Chemical Information and Modeling* 53 (2013), 867–878 (cit. on p. 1).

[21] R. Mazumder, T. Hastie, and R. Tibshirani. „Spectral regularization algorithms for learning large incomplete matrices“. In: *Journal of Machine Learning Research*. 2010, pp. 2287–2322 (cit. on p. 6).

[22] W. S. Mcculloch and W. Pitts. „A logical calculus of the ideas immanent in nervous activity“. In: *Bulletin of Mathematical Biophysic* 5 (1943), pp. 115–133 (cit. on p. 13).

[23] T. M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, Mar. 1997 (cit. on p. 10).

[24] NM O’Boyle, M Banck, CA James, et al. „Open Babel: An open chemical toolbox“. In: *Journal of Cheminformatics* 3 (2011), p. 33 (cit. on p. 4).

[25] M. Ojala and G. C. Garriga. „Permutation Tests for Studying Classifier Performance“. In: *Journal of Machine Learning Research* 11 (Aug. 2010), pp. 1833–1863 (cit. on p. 25).

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 19).

[27] J. R. Quinlan. „Induction of Decision Trees“. In: *Machine Learning* 1 (1986), pp. 81–106 (cit. on pp. 9, 10).

[28] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993 (cit. on p. 10).

[29]J. Schmidhuber. „Deep Learning in Neural Networks: An Overview“. In: *Neural Networks* 61 (2015). Published online 2014; based on TR arXiv:1404.7828 [cs.NE], pp. 85–117 (cit. on p. 13).

[30]R.P. Schwarzenbach, P.M. Gschwend, and D.M. Imboden. *Environmental Organic Chemistry*. Wiley, 2005 (cit. on p. 6).

[31]Leo Breiman Statistics and L. Breiman. „Random Forests“. In: *Machine Learning*. 2001, pp. 5–32 (cit. on p. 11).

[32]S. Van Der Walt, S. C. Colbert, and G. Varoquaux. „The NumPy array: a structure for efficient numerical computation“. In: *Computing in Science and Engineering* 13.2 (Mar. 2011), pp. 22–30 (cit. on p. 19).

[33]P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. „Extracting and Composing Robust Features with Denoising Autoencoders“. In: *Proceedings of the 25th International Conference on Machine Learning*. ICML ’08. Helsinki, Finland: ACM, 2008, pp. 1096–1103 (cit. on p. 15).

[34]P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. „Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion“. In: *Journal of Machine Learning Research* 11 (Dec. 2010), pp. 3371–3408 (cit. on pp. 15, 16).

[35]S. Vorberg and I. V. Tetko. „Modeling the Biodegradability of Chemical Compounds Using the Online CHEmical Modeling Environment (OCHEM)“. In: *Molecular Informatics* 33 (2014), 73–85 (cit. on p. 1).

[36]J. Wicker, T. Lorsbach, M. Gütlein, et al. „enviPath - The Environmental Contaminant Biotransformation Pathway Resource“. In: *Nucleic Acid Research* 44.D1 (Jan. 4, 2016). Ed. by Michael Galperin, pp. D502–D508. published (cit. on p. 3).

[37]C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. „Understanding deep learning requires rethinking generalization“. In: *Computing Research Repository* abs/1611.03530 (2016) (cit. on pp. 33, 37).

[38]Z. Zhang. „Multiple imputation with multivariate imputation by chained equation (MICE) package“. In: *Annals of Translational Medicine* 4.2 (2016) (cit. on p. 6).

List of Figures

List of Tables