University of Antwerp

# Chemo-informatics and computational drug design

Prof. Dr. Hans De Winter

University of Antwerp

Campus Drie Eiken, Building A

Universiteitsplein 1, 2610 Wilrijk, Belgium

Gefinancierd door de Europese Unie

NextGenerationEU

# Chapter 5. Quantitative structure-activity relationship models

## 1. Some general concepts

### 1.1. Model building strategy

Apart from similarity searches, diversity analysis and clustering, chemo-informatics approaches are often used to develop sophisticated models that are capable to predict biological activities for molecules that have never been tested before (QSAR: quantitative structure-activity relationship). The methods that underly these models are called "machine learning" methods, and may vary from simple linear regression approaches up to the more sophisticated deep-learning and so-called "artificial intelligence" methods.

The purpose of each developed model is to predict, given a molecular structure as input, some kind of property from that molecular structure. If this property is a biological activity, then the model is called a QSAR model. If the property is rather a physicochemical property, such as aqueous solubility, then one talks about QSPR models (quantitative strutucre-property relationship). In order to build a QSAR or QSPR model, each model has to be trained using some existing property data of interest, and the corresponding molecular structures. A model building process generally consists in three stages: 1) *model generation*, 2) *model validation*, and 3) *production run* (Figure 30).

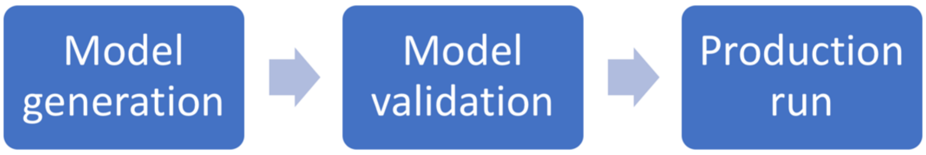

*Figure 30. The three stages in a model building experiment.*

Once a suitable model has been generated, its predictive power and quality should be validated. This validation is done in the second phase, which is called the validation phase. If it turns out that the quality of the model is not sufficient enough, then a new model has to be generated and again validated, until the final model reaches the desired quality. In this case, the actual production stage may start. Validation is a crucial aspect of any virtual screening process. It is the process by which the reliability and relevance of a procedure are established for a specific purpose. The different validation methods and metrics will be discussed in the following sections.

A QSAR model is nothing more than a mathematical description that links whatever kind of chemical descriptor (such as a molecular fingerprint) to the desired property that one want to model. To illustrate this concept, consider a simple linear regression model in which property *y* (for example the molecular lipophilicity) is linked to descriptor *x* (for example the molecular weight of the molecule):

$$y = ax + b$$

In order to be able to link *y* to *x*, coefficients *a* and *b* need to be known, and this is achieved by training the linear regression equation with a set of trainingsdata that contain many y and x as examples. Using simple mathematics, it is then possible to derive *a* and b in order to generate a linear regression model.

This example may be somewhat too simple in real situations, but it illustrates well the concept of model building within the world of chemo-informatics and computational drug design. In practice, one is often interested in the prediction of biological activities (*y*) using chemical fingerprints as input descriptor (*x*). Instead of the simple linear regression model as given in this example, more sophisticated methods are used to correlated *y* to *x*. In this section, we will cover some of these methods that are used to build models, but we will start by describing the basics that are needed to validate the quality of these models using performance metrics.

## 1.2. Classification and regression models

A model can either indicate whether a molecule is predicted to be 'active' or 'inactive' (1 versus 0), or it can return a predicted binding affinity value as a real number. In the former case, the returned data type is binary (or class), while in the latter case the returned data type is continuous (Figure 31). Pharmacophore searches (see later) are often binary classification methods, as the molecule may fit the pharmacophore or not. Docking screens on the contrary often return a scoring value and are therefore continuous models.

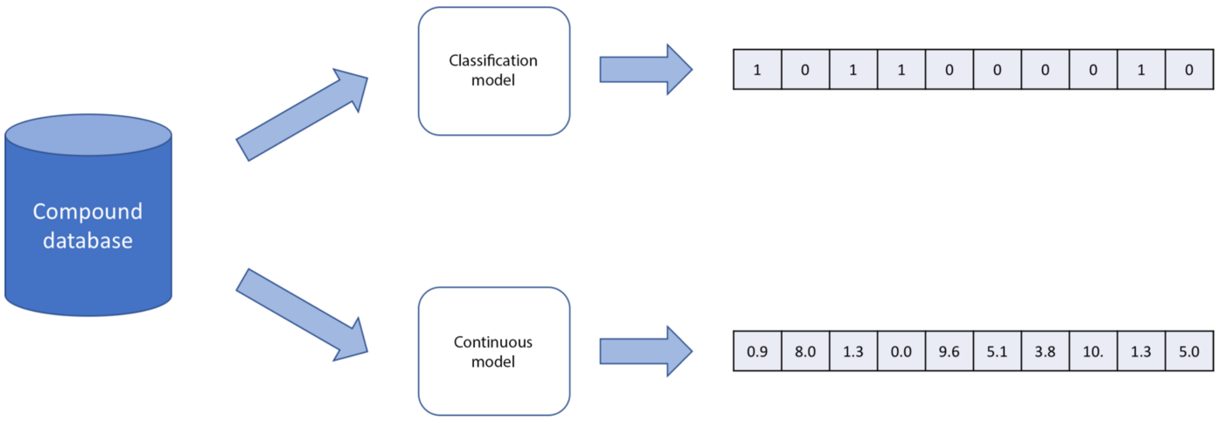

*Figure 31. Two types of data generated by virtual screening methods: classification models (binary models) produce binary data ('active' or 'inactive', '1' or '0'), while continuous models produce continuous data (for example the value calculated by a docking scoring function, or the biological activity expressed in pIC$_{50}$ units).*

When the model performs a classification, one calls this model a classification model. Such a model is able to predict the class of the unknown datapoint (for example a new molecule) to which this datapoint belongs (for example, class "ACTIVE" or class "INACTIVE"). On the other side of the spectrum, regression models are able to predict a continuous property value for the unknown datapoint, such as a biological activity or aqueous solubility.

## 1.3. Supervised and unsupervised learning

Within machine learning, there are two basic approaches: supervised learning and unsupervised learning. The main difference is that in supervised learning one uses labeled data to build models, while this is not the case for unsupervised learning.

### *Supervised learning*

Supervised learning is a machine learning approach that is defined by its use of labeled datasets. In supervised learning, the model is trained using data which is labeled. It means some data is already tagged with the correct answer. It can be compared to learning which takes place in the presence of a supervisor or a teacher. As an example of such a labeled dataset, consider a set of compounds of which the biological activities for a given target is known. In this case, the compounds can be considered to be "labeled" with these affinity data, and the model can be trained to predict the affinities of novel compounds.

Supervised learning can be separated into two types of problems when data mining: classification and regression:

- *Classification* problems use an algorithm to accurately assign test data into specific categories, such as separating active compounds from inactive ones. Classification problems are solved using classification models (see the section before). Linear classifiers, support vector machines, decision trees, neural networks and random forest are all examples of classification algorithms. In this course, we will mainly focus on decision trees and random forest methods.
- *Regression* is another type of supervised learning method that models the relationship between the dependent (the observation one wants to predict, such as the biological activity) and independent variables (for example the molecular fingerprint bits). Regression problems are solved using regression models. Such regression models are helpful for predicting numerical values based on different data

points. Some popular regression algorithms are linear regression, some flavors of neural networks, logistic regression and polynomial regression. In this course, we will mainly examplify linear regression.

### *Unsupervised learning*

Unsupervised learning uses machine learning algorithms to analyze and cluster unlabeled data sets. These algorithms discover hidden patterns in the data itself without any assigned labels.

Unsupervised learning models are used for two main tasks: clustering and dimensionality reduction.

- *Clustering* is a data mining technique for grouping unlabeled data based on their similarities or differences. For example, *K*-means clustering algorithms assign similar data points into groups, where the *K* value represents the size of the grouping and granularity. In this course, we have already covered hierarchical and non-hierarchical clustering methods (see before).
- *Dimensionality reduction* is a learning technique used when the number of features (or dimensions) in a given dataset is too high. It reduces the number of data inputs to a manageable size while also preserving the data integrity. Often, this technique is used in the preprocessing data stage, hence before the actual model building step takes place. A typical example of dimensionality reduction is principle component analysis (PCA). In this course, we will not cover any of the dimensionality reduction methods.

## 1.4. Descriptive and predictive analytics

Apart from the model types and learning methods, we can distinguish between two types of data analytics:

- Descriptive analytics
- Predictive analytics

### *Descriptive analytics*

In descriptive analytics, one tries to answer the question of 'what happened?' In descriptive analytics, one looks at the data and tries to obtain a detailed description of the data itself. In descriptive analytics one mainly looks at the variables separately. Example of descriptive analytics algorithms are the creation of tables with frequencies or percentages of the different values or groups of values, summary statistics like the mean (for continuous variables) or the median. These two well known summary statistics tell us something about the "middle" of the data, but there are also summary statistics regarding the spread (i.e. are different data points close to each other in value or not) like the variance or interquartile range. Graphical representations of the data play a key role as they give insight into the complete distribution of the variable. These are often closely related to the summary statistics, like in the case of a boxplot. The most commonly known graphical representations are bar charts for all non-continuous data and histograms for the continuous variables. Special interest goes to two odd kinds of values. There often are outliers, cases which are so far removed from the rest of the data that they might be mistakes. If not properly identified if these values are indeed valid they can have a disproportionately big impact on all further analyses. Apart from outliers one also needs to look out for missing values. In case of missing values, the issue must be resolved before one can continue with the analysis. In any given case missing values should not plainly be converted to a zero as this would of course result in incorrect statistics and conclusions.

### *Predictive analytics*

Prediction is main focus of all methods that are covered in this course, as it tells us what would happen given new data based on data from the past. In the case of computational drug design, models are derived that allows the researcher to look for other compounds with the desired property. Linear regression is already a form of prediction. Confidence intervals are often used here as they give an expected range rather than only a single value.

## 2. Building QSAR models

There exist many machine learning approaches that can be used to generate models with. Many machine learning approaches can be used to generate both classification or continuous models. In this section, a number of the more commonly used methods are presented and their use explained. We will not focus on the mathematical implementations; rather we will highlight some important features of each of the methods and explain the advantages and disadvantages.

For all machine learning methods in this course, the common and open source package scikit-learn is used to demonstrate the concepts of QSAR model building. It is by default available from within Google Colab, so there is no need for additional installations unless you need scikit-learn on your own local system. In that case, installation instructions are provided on the scikit-learn website.

In order to be able to build machine learning models, there are two important data sources that should be available:

- The molecules in a computer-readable format. Often, standard 1024- or 20480-bit fingerprints are used for this, but other representations, such as the logP, molecular weight, number of atoms, are also possible.
- The data that should be modeled. In the case of building QSAR models, this is often some kind of biological activity that one wants to model (for example, represented as $IC_{50}$ or $pIC_{50}$ values). In the case of QSPR models, this can be the physicochemical property such as the to-be-modeled aqeous solubility (for example, expressed in g/L).

The combination of molecules and corresponding data is called the "training set". The quality of this training set is of utmost importance for the quality of the generated model. When the datapoints (for example, the biological activities) are of lesser quality, then the corresponding model will also suffer from lesser quality.

Molecular structures cannot be used by the computer as such. Rather, they need to be converted in a format that the computer can understand. Very often, standard fingerprints, such as the standard Daylight-, Morgan-fingerprint or MACCS-keys are used. Once all molecules are converted in such a representation, this training set is then fed into the machine learning application to calculate a model.

As an example of a typical training set, consider the following snippet from a file with compounds that were tested on their potential DPP4-inhibition capabilities:

```
ACTIVE      CN[C@@H](C1CCC(CC1)NS(=O)(=O)c2ccc(F)cc2F)C(=O)N3CC[C@H](F)C3
ACTIVE      CN(C)C(=O)[C@H]([C@H](N)C(=O)N1CC[C@H](F)C1)C2CCC(CC2)N(C)C(=O)c3cccc(F)c3
ACTIVE      C[C@@H](B(O)O)N1CC[C@H](N)C1=O
ACTIVE      CN(C)C(=O)[C@H]([C@H](N)C(=O)N1CC[C@H](F)C1)C2CCC(CC2)NS(=O)(=O)c3ccc(F)cc3F
ACTIVE      CN(C)C(=O)[C@H]([C@H](N)C(=O)N1CC[C@H](F)C1)c2ccc(cc2)N(C)C(=O)c3ccc(F)cc3
...
INACTIVE    Cn1nnnc1SCc1ccc(C(=O)Nc2ccccc2F)cc1
INACTIVE    O=C1/C(=C/c2ccccc2Br)Oc2cc(OCc3ccc(F)cc3)ccc21
INACTIVE    Cc1nn(C(C)C(=O)NCc2ccccc2Cl)c2ccccc12
INACTIVE    CCCOc1ccccc1C(=O)Nc1ccc(NC(=O)C(C)C)cc1
INACTIVE    OCCNc1nc(Nc2ccccc2O)nc2ccccc12
...
```

The first column indicates whether the compound is an inhibitor of DPP4 ("`ACTIVE`" when $pIC_{50} > 4$), or not ("`INACTIVE`"). Each compound is represented by its SMILES representation. As this dataset contains only `ACTIVE` or `INACTIVE` as classes, it can be used to build classification models but it is not suitable to build continuous models from. Other datasets can include the actual $pIC_{50}$ values as property, and these datasets can then be used to build regression models.

## 2.1. Linear regression models

In linear regression, the target value (*e.g.* the property that needs to be predicted) is modeled as a linear combination of the molecular features (*e.g.* the individual fingerprint bits). Linear regression is probably the most simple of all machine learning models, but it offers interpretability and ease of calculation.

Linear regression fits a linear model with coefficients $w = (w_1,\dots,w_p)$ to minimize the residual sum of squares between the observed targets (properties) in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} \|Xw - y\|_2^2$$

as it is common to use the sum of squared errors $\|\varepsilon\|_2^2$ as a measure to minimize.

Good practices require the use of a training set and a test set. This is done by randomly diving the entire dataset into these two sets using a certain fraction, for example 70% of the dataset becomes the training set and the remaining 30% becomes the test set. The model is then trained using the data in the training set, and the resulting model is subsequently validated using the test set. A common metric to evaluate a linear regression model is the mean squared error of the test set, defined as:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

with n the number of datapoints in the test set, $\widehat{Y}_i$ the predicted value of datapoint *i*, and $Y_i$ the real value of datapoint *i*. In other words, the *MSE* is the mean of the squares of the differences between real values and predictions. If this difference is close to 0, then the *MSE* will also be close to 0.

A second obvious validation of the generated model is the visual inspection of the plot that scatters the predicted values against the real values. There should be a clear correlation between both, as is shown in Figure 32. If this correlation is absent, then the generated model has no predictive value and is of little use in a typical QSAR setting.

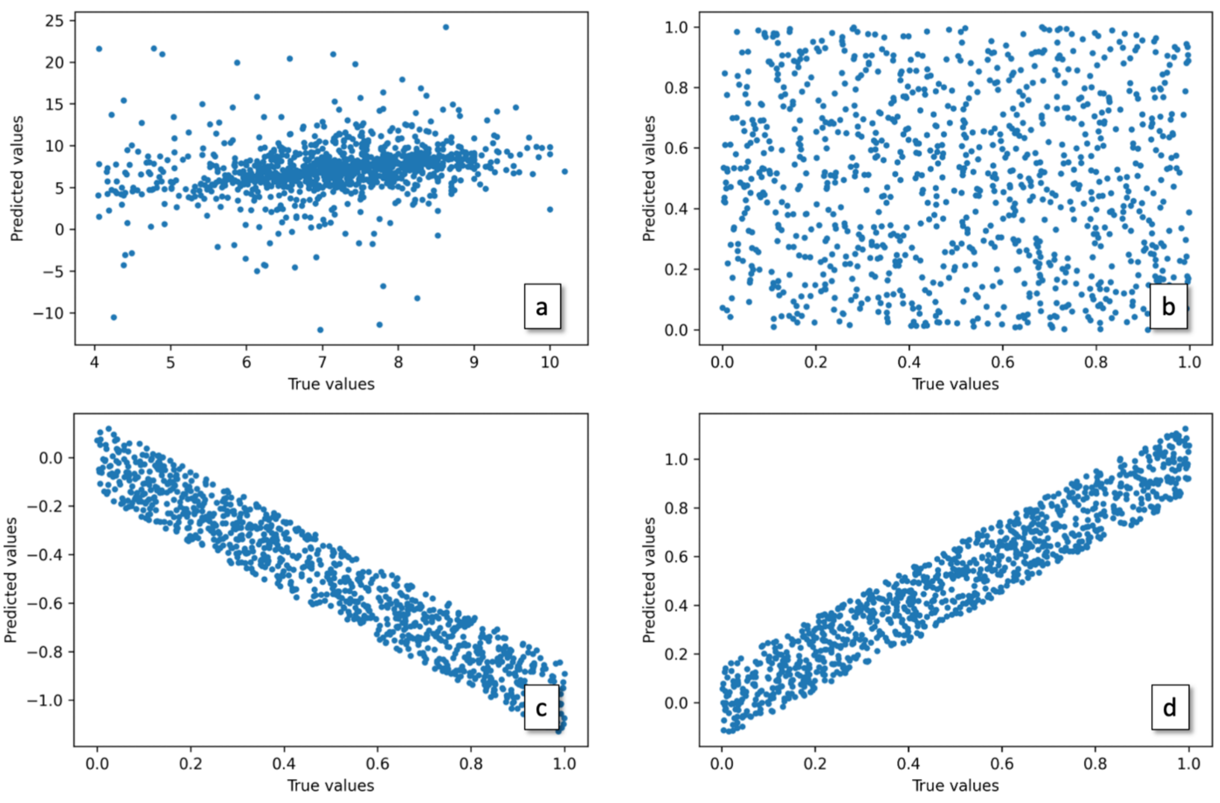

*Figure 32. Examples of different correlation plots. (a) Example of a weak correlation with large variations in the predicted values. (b) No correlation at all. (c) Negative but good correlation between the true and predicted datapoints, indicating a good predictive model. (d) Positive and good correlation between the true and predicted values.*

To illustrate the process of QSAR model building, we will consider an example in which the inhibitory activity of chemical compounds against the dipeptidyl-peptidase IV (DPP4) is modeled using a linear regression model. The ChEMBL database was used to extract all small molecule compounds having biological activity data against DPP4 (stored as $p\text{IC}_{50}$ values). In total, 3,858 of such compounds were retrieved, and their chemical structures (stored as SMILES strings) were converted into Daylight fingerprints consisting of 2,048 bits per fingerprint:

```
url = "https://raw.githubusercontent.com/UAMCAntwerpen/2040FBDBIC/main/dpp4.pIC50.txt"
data = requests.get(url).text.split("\n")
fps = []
pic50 = []
for d in data:
  fields = d.split()
  if len(fields) < 1: continue
  pic50.append(float(fields[1]))
  mol = Chem.MolFromSmiles(fields[0])
  fp = np.zeros((0,), dtype=np.int8)
  DataStructs.ConvertToNumpyArray(Chem.RDKFingerprint(mol), fp)
  fps.append(fp)
```

Prior to building a model, the dataset is first split into a training set (70% of the compounds) and a test set (30%). The training set is used to train the model, while the test set is kept aside and used later on to validate the generated model:

```
pic50_train, pic50_test, fps_train, fps_test = train_test_split(pic50, fps, test_size=0.3)
model = linear_model.LinearRegression()
model.fit(fps_train, pic50_train)
```

Finally, the generated linear regression model is validated with the test set. For this purpose, the generated model is applied on the fingerprints of the test set, and the hence generated $p\text{IC}_{50}$ values (the predictions) are compared with the true $p\text{IC}_{50}$ values as given by the test set. This comparison is done with the *MSE* metric:

```
pic50_pred = model.predict(fps_test)
print("MSE = ", mean_squared_error(pic50_test, pic50_pred))
```

The calculated *MSE* is 9.3, meaning that the average squared difference between the real and predicted $p\text{IC}_{50}$ values is around 9.3. Given that the $p\text{IC}_{50}$ values in the dataset range between 4-10, a *MSE* of 9.3 is therefore a large error, indicating a bad predictive model. This is also obvious when plotting the predicted valaues against the corresponding real values:

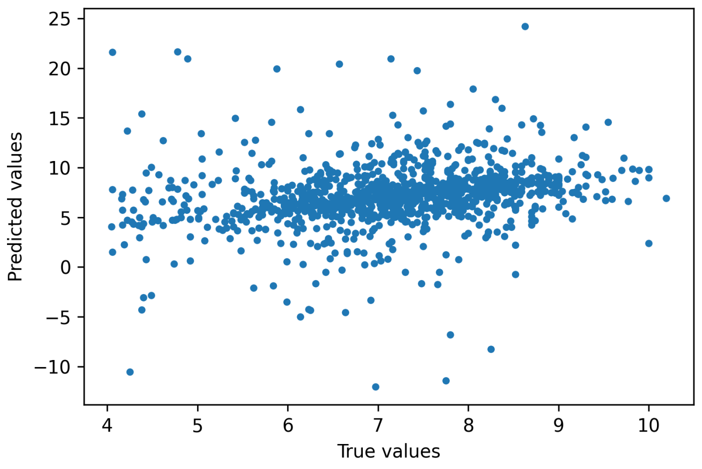

*Figure 33. Correlation plot between the true and predicted pIC50 values of a linear regression model of DPP4 inhibition. The lack of obvious correlation is indicative of a bad predictive model, also confirmed by the MSE which is around 9.3.*

*Linear regression models are models of continuous datasets, like the prediction of biological activity data such as $IC_{50}$ values*

Linear regression models are typically used to predict numerical properties such as biological activities or physicochemical properties. However, **logistic regression** is a regression analysis technique that can be applied on **classification problems**. Please consult this Youtube video if you want to learn more about this useful technique in machine learning (8'47" in length).

## 2.2. Neural network models

Neural networks form the basis of deep learning and artificial intelligence approaches. It has undergone a huge transformation with the recent availability of powerful graphical processing units (GPU's) which are optimally suited to perform calculations needed for neural networks. Since deep learning and neural networks are available in many flavors, going from recursive deep learning, convolutional deep learning and reinforcement deep learning, we will only very briefly explain the basic concepts of neural networks in this section.

A neuron, in the context of neural networks, is a fancy name for a "function". A function takes some input *x*, applies some logic *f()*, and outputs the result *f(x)*:

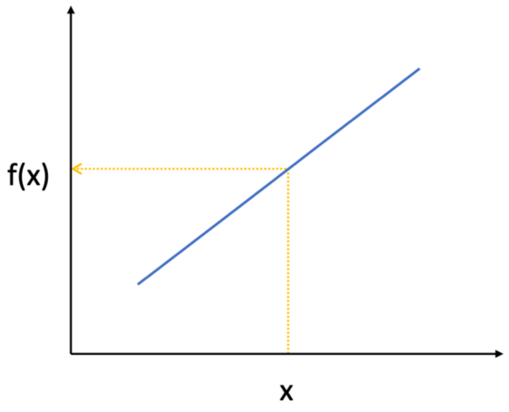

*Figure 34. A neuron represented as a simple mathematical function, taking some input x and returning a result f(x).*

Hence, since that a neuron is nothing more than a function, a neural network is nothing more than a network of functions:

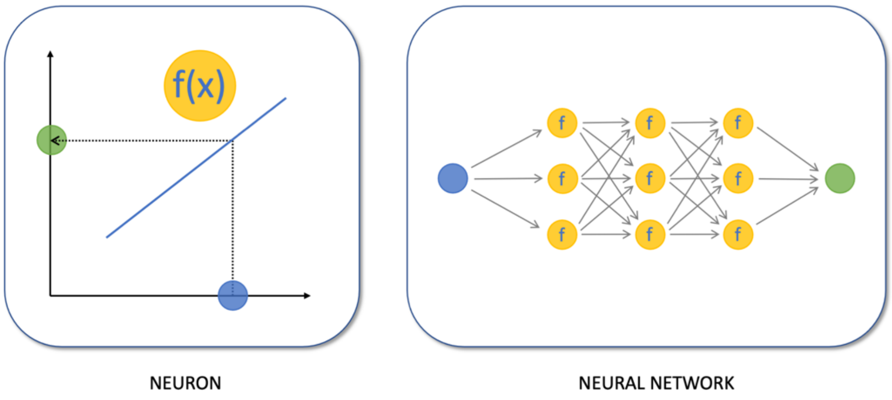

*Figure 35. Relation between a neuron and a neural network. A neural network consists of a network of functions (neurons), each function is linked by its input and output to the other functions. Neural networks mainly differ in 1) the arrangement (connectivities) between the functions and 2) the type of function f(x).*

As can be seen from Figure 35, each neuron in the neural network is connected to a number of other neurons. Each neuron in a given column of neurons receives its input from each of the neurons in the column on the left side, and sends its output, after transformation with f(x), to each neuron of the column on its right side. In Figure

36, the connection between a single neuron and its *n* preceding neurons is highlighted. The figure depicts a neuron connected with *n* other neurons and thus receives *n* inputs ($x_1$, $x_2$, $x_3$, …, $x_n$). Each input signal is weighted with a specific weight for that particular connection. This configuration is called a perceptron. All the inputs are individually weighted, added together and passed into the activation function. There are many different types of activation functions but one of the simplest is the step function. A step function will typically output 1 if the input is larger than a certain threshold, otherwise it will output a 0. The inputs ($x_1$, $x_2$, $x_3$, …, $x_n$) and weights ($w_1$, $w_2$, $w_3$, …, $xw_n$) are real numbers and can be positive or negative. Hence, a perceptron consists of weights, a summation processor and an activation function.

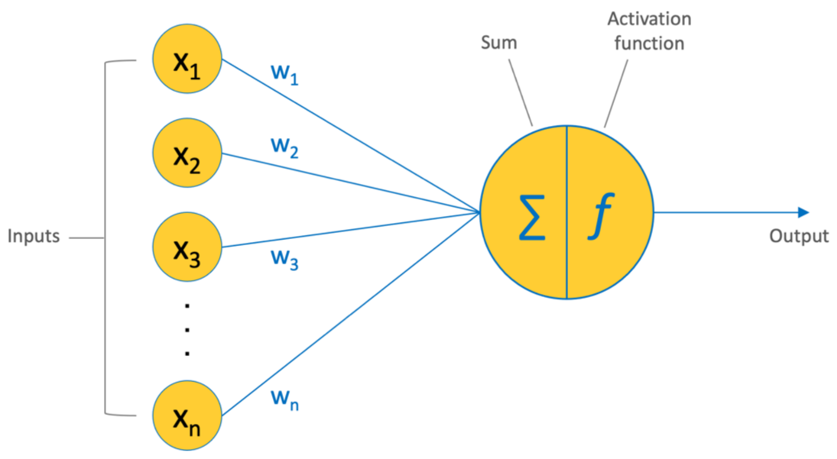

*Figure 36. A single neuron connected to n other neurons. The neuron receives n inputs ($x_1$, $x_2$, $x_3$, …, $x_n$). Each input is weighted with a corresponding weight ($w_1$, $w_2$, $w_3$, …, $w_n$). Such a configuration is called a perceptron.*

The training of a neural network is nothing more than the optimisation of the network so that the difference between the true and predicted values is minimised. Optimisation of the network includes the optimisation of the connectivities between the neurons (weights of the connections and the type of connectivities), the optimisation of the function parameters and the functions themselves, and other stuff which is depending on the flavor of deep learning network. How this learning process works is beyond the scope of this course, but to learn more about this Youtube video is more than worth looking at (length is 11'18").

Neural networks can be used both on continuous and classification datasets, like the prediction of numerical properties (for example, $IC_{50}$) or the prediction of a certain class (for example, "ACTIVE" versus "INACTIVE")

## 2.3. Random forest models

The random forest classifier is a useful machine learning tool that fits a number of decition tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. It is mainly used for classification applications, but there exist flavors of the random forest technology that can be used in regression applications as well. In this section, it will be explained how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

**Decision trees** as they are the building blocks of random forest models. The concept of a decision tree is quite straigthforward and is best illustrated using an example (Figure 37). The dataset consists of different shapes in different colors. There are two squares in red, five circles of which one is red and four are colored blue. As such, the features are 'shape' and 'color'. So how can we do this?

In first instance, color seems like a pretty obvious feature to split by as the majority of the shapes are colored blue. So we can use the question, 'Is color red?' to split our first node. You can think of a node in a tree as the point where the path splits into two - observations that meet the criteria go down the 'YES' branch and ones that don't go down the ‘NO’ branch. At this point, the ‘NO’ branch (the blues) contains nothing but circles so we are done there, but our ‘YES’branch can still be split further. This can be done based on the second feature and ask “Is it a square?” to make a second split. The two squares go down the ‘YES’ subbranch and the circle goes down the ‘NO’ subbranch and we are all done.

In this example, our decision tree was able to use the two features to split up the data perfectly. Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same. At each node, it will ask: “*what feature will allow me to split the observations at hand in a way that the resulting groups are as different from each other as possible (and the members of each resulting subgroup are as similar to each other as possible)?*”

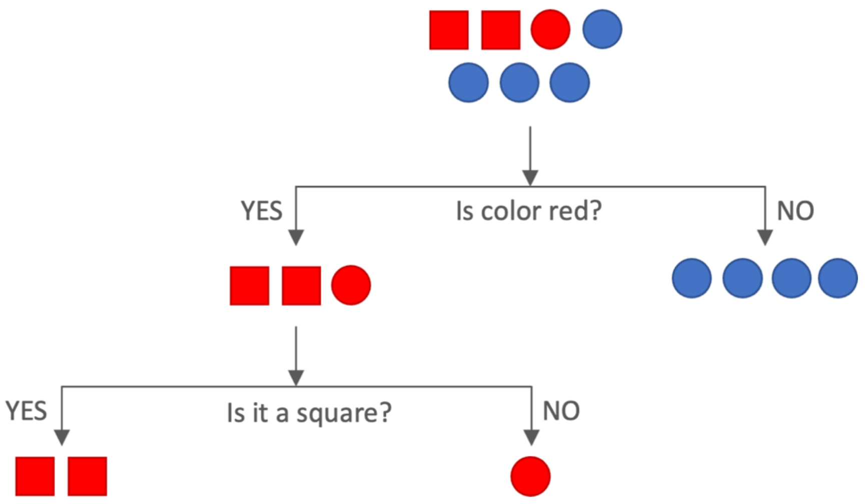

*Figure 37. Example of a decision tree that was modeled to separate three different objects based on their shape and color.*

Training (building) a decision tree comes down to finding the best cutoff parameters to separate each dataset onto two separate sets. Obviously, a decision tree only takes into account the provided parameters with each data sample. For example, in the case of molecules characterised by their 1024-bits fingerprints, a possible question could be: “Is the bit at position 245 a 1 or a 0?”

**Random forests**, like its name implies, consist of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes the prediction of the random forest (see Figure 38). The fundamental concept behind random forest is based on “the wisdom of crowds”. In data science terms, the reason why the random forest models work so well is that a large number of uncorrelated models (the individual decision trees) that operate as an ensemble will outperform any of the individual constituent models.

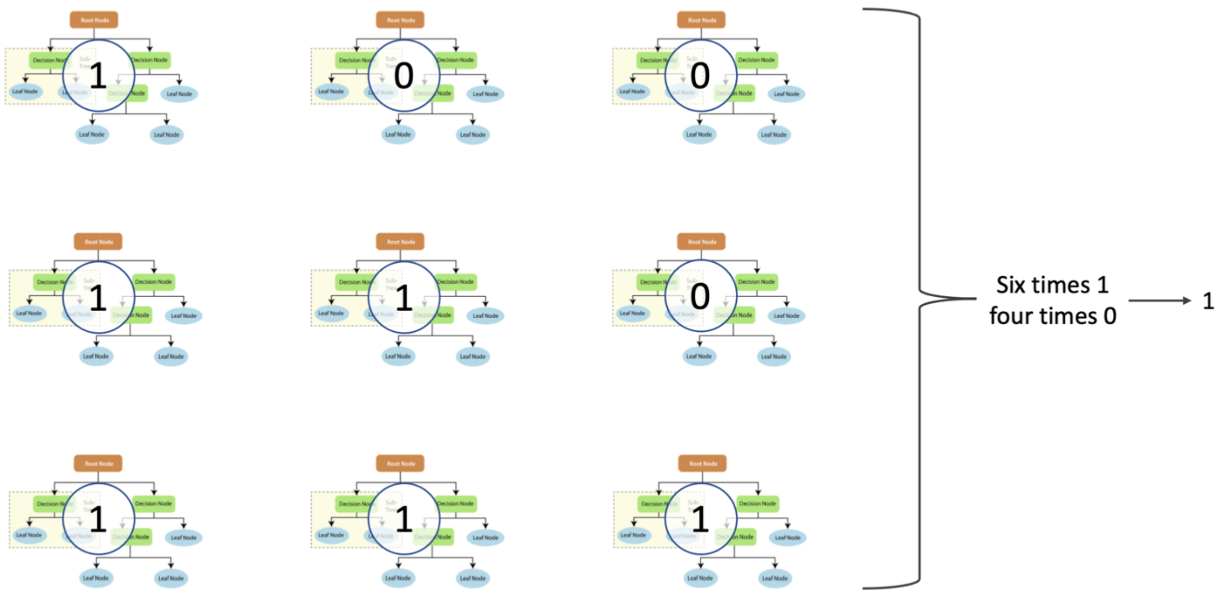

*Figure 38. Illustration of a random forest model constituted out of nine uncorrelated decision trees. Six decision trees predict the value to be 1, and four trees predict a 1. Hence, the prediction made by the random forest is 1.*

The low correlation between the individual models is the key concept behind the success of random forests. Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this effect is that the trees protect each other from their individual errors. While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

- There needs to be some actual signal in our features so that models built using those features do better than random guessing.
- The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Random forest models ae most often used for classification problems, but there are random forest flavors that are designed to handle continuous datasets

## 3. Validation

### 3.1. Validation of classification models

There exist many metrics to define the quality of a model, and the actual choice of a metric depends on the question one wants to be answered. Sometimes one needs a model that is capable of selecting *all* active compounds that are contained in the database, but at the cost of having to test many compounds from that database. At other times, one might just be interested to identify only a few active compounds from the database, with the risk of missing out many others. In order to quantify these properties, a number of performance definitions and metrics have been described.

### The confusion matrix

The confusion matrix is one of the most intuitive and easiest metrics used for finding the correctness and accuracy of the model. It is used for classification problems where the output can be of two or more types of classes, for example a compound is predicted by the model to be *active* (1) or *inactive* (0).[2]

The confusion matrix in itself is not a performance measure as such, but almost all of the performance metrics are based on confusion matrix and the numbers inside it (Figure 39).

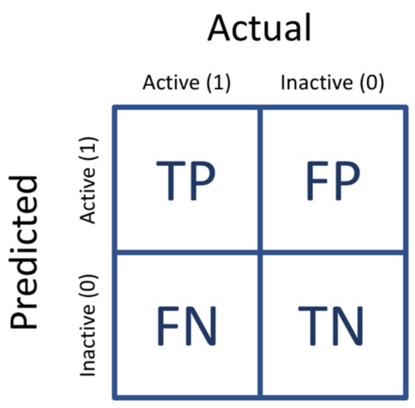

*Figure 39. The confusion matrix is a table with two dimensions ('Actual' and 'Predicted'), and sets of 'classes' in both dimensions. The 'actual' classifications are columns and the 'predicted' ones are rows. TP: true positives, TN: true negatives, FP: false positives, FN: false negatives.*

**True positives (TP)** – True positives are the compounds when the real measured biological activity of the compound is 'active' and the predicted activity also 'active'.

**True negatives (TN)** – True negatives are the cases when the real measured biological activity of the compound is 'inactive' and the predicted activity also 'inactive'.

**False positives (FP)** – False positives are compounds for which the measured biological activity is 'inactive' but the predicted activity is 'active'.

**False negatives (TF)** – False negatives are the cases for which the real biological activity is 'active' but the predicted activity is 'inactive'.

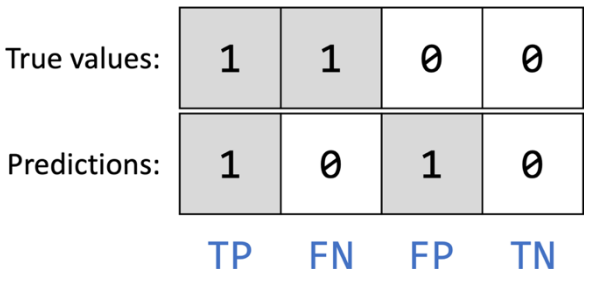

*Figure 40. Illustration of the concepts of true positives (TP), false positives (FP), true negatives (TN) and false negatives (FN).*

The ideal model should give zero false positives and zero false negatives, but due to the inherent error in all models this goal is never achieved. We know that there will be some error associated with every model that is used for predicting the true class of the target compound. This will result in false positives and false negatives. There is no hard rule that says what should be minimized in all the situations, and depends on the business needs and the context of the problem one is trying to solve. Based on that, we want our model to minimize either the false positives or the false negatives:

1. Minimizing false positives. If one wants to mine a compound database for compounds that are active against a particular therapeutic target, but one has limited resources to physically purchase these

[2] In case of a continuous model such as the scoring function of a docking experiment, these continuous data first have to be transformed into binary data as explained later.

compounds so that only a limited number can be acquired and biologically tested, then one needs a model that minimizes false positives. By minimizing false positives, the researcher can be relatively 'sure' that any compound that is predicted by the model as 'active' is actually also active, hence limiting waste of financial resources on the purchase of inactive compounds. The drawback of a such model is that many 'active' compounds in the database will not be picked up, hence potentially missing out some real good active compounds.

2. Minimizing false negatives. A model that minimizes false positives will pick up 'all' active compounds from the database, however at the cost of having to experimentally screen a large number of compounds including many false positives. This model is useful if one expects that the database will not contain many active compounds and one wants to retrieve as much as possible of these active ones.

### *True positive rate: sensitivity or recall*

The true positive rate (also called *sensitivity*, *recall*, *hit rate* or *power*) is the probability that an actual positive value will be predicted as being positive by the model. A model with a large *TPR* is capable of retrieving the majority of the real positives in the database (small number of *FN*), however at the cost of retreiving also many false positives (Figure 42a). In this context, a model that predicts all compounds to be active (irrespective whether they really are active) has a sensitivity of 1. As such a model is useless in practical applications, a second metric, in addition to the sensitivity metric, is warranted for model validation. An example of a metric useful to include could be the *FPR* (*vide infra*). By combining the *TPR* and *FPR* metrics one obtains the *accuracy* metric (*vide infra*): it optimizes the *TPR* and minimizes the *FPR* (Figure 41).

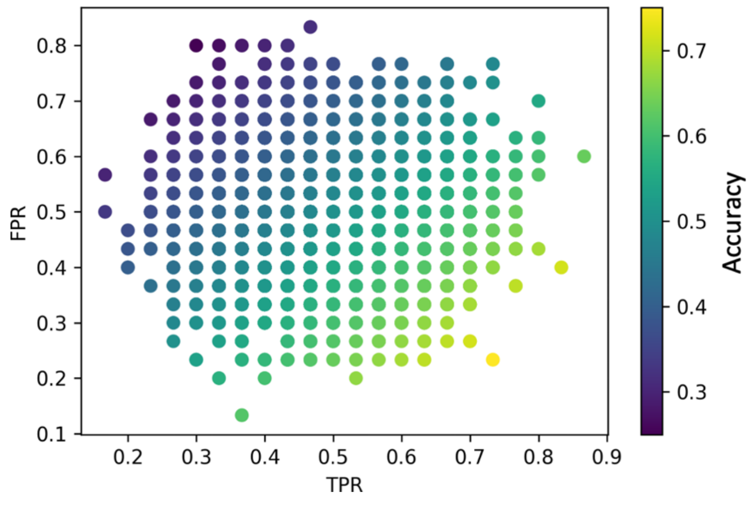

*Figure 41. Relation between the true positive rate (TPR), false positive rate (FPR) and accuracy (color-coded; large accuracy value are yellow, small accuracy values are blue). Figure was created by generating 10,000 random models and calculating the appropriate metrics from these models. In a typical virtual screening application, a model with a large TPR and a small FPR is often desired (lower-right corner). Such models have a high accuracy (yellow dots).*

The recall is a measure that tells what proportion of real active compounds are predicted by the algorithm as active. Since the real actives can be defined as *TP* and *FN*, and the compounds predicted by the model as actives equals to *TP*, sensitivity or recall can be written as:

$$Sensitivity = recall = \frac{TP}{TP + FN} = TPR$$

### *True negative rate: specificity*

The true negative rate (*TNR*, also called *specificity*) is the probability that an actual negative value will be predicted as being negative. A model with a high specificity is capable of retrieving the majority of the real negatives in the database, however at the cost of retrieving also many false negatives (Figure 42b).

As specificity is the measure that tells what proportion of the compounds that were predicted as being inactives are really inactive, its value depends in the the actual negatives (equalling to *FP* + *TN*) and the compounds predicted to be inactive (equalling to *TN*). Hence, specificity becomes:

$$Specificity = \frac{TN}{TN + FP} = TNR$$

Specificity is the exact opposite of sensitivity. If a model is altered to increase sensitivity, then the specificity of that model will decrease, and *vice versa*.

### *False positive rate: fall-out*

The false positive rate (*FPR*) is the probability that a false alarm will be generated by the model. Models that are characterised with a high *FPR* will suggest a large number of positive hits, while these predictions will prove to be negative once these are tested in the lab. The false positive rate is calculated from following equation:

$$Fall\ out = \frac{FP}{FP + TN} = FPR$$

The false positive rate can also be defined as:

$$FPR = 1 - specificity$$

### *False negative rate: miss rate*

The false negative rate (*FNR*, also called the *miss rate*) is the probability that a true positive hit will be missed by the model. Hence, if it is crucial to retrieve all hits from the database, then a model with a small *FNR* should be used. The false negative rate is calculated from following equation:

$$FNR = \frac{FN}{FN + TP}$$

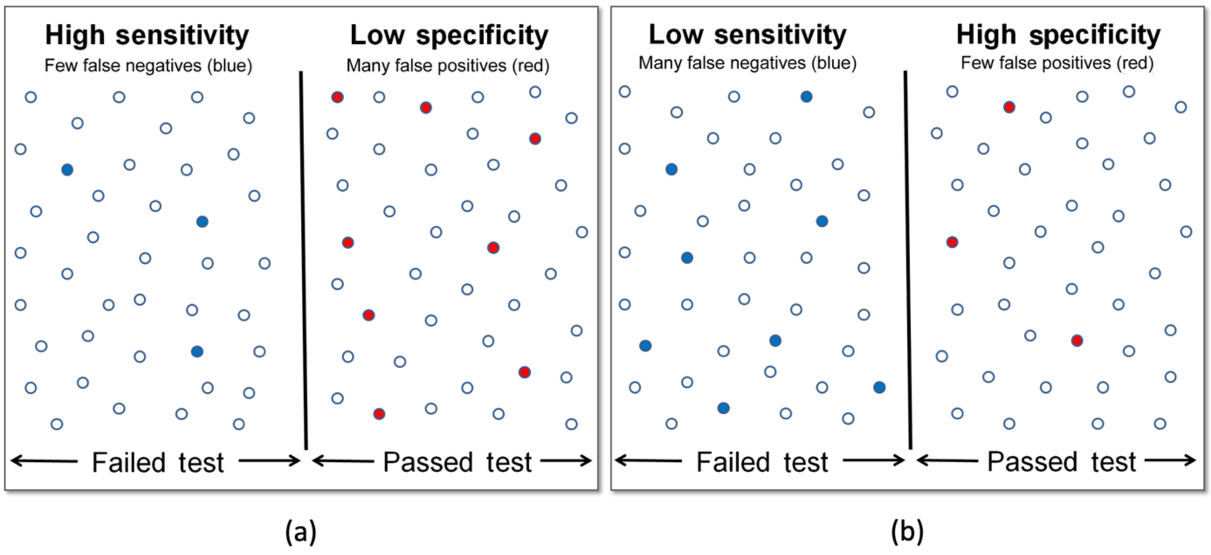

*Figure 42. Trade-off between specificity (TNR) and sensitivity (TPR). Panel (a) shows the results for a model with high sensitivity and low specificity. This model will generate few false negatives at the cost of retrieving a large fraction of false positives. Panel (b) illustrates a model characterised by a low sensitivity and a high specificity. Such a model retrieves only few false positives but this comes at a cost of retrieving many false negatives. Figure adapted from https://commons.wikimedia.org/w/index.php?curid=14502690.*

### *Accuracy and precision*

An accurate test is a test in which systematic errors are reduced, while a precise test is a test in which random error are limited (Figure 43).

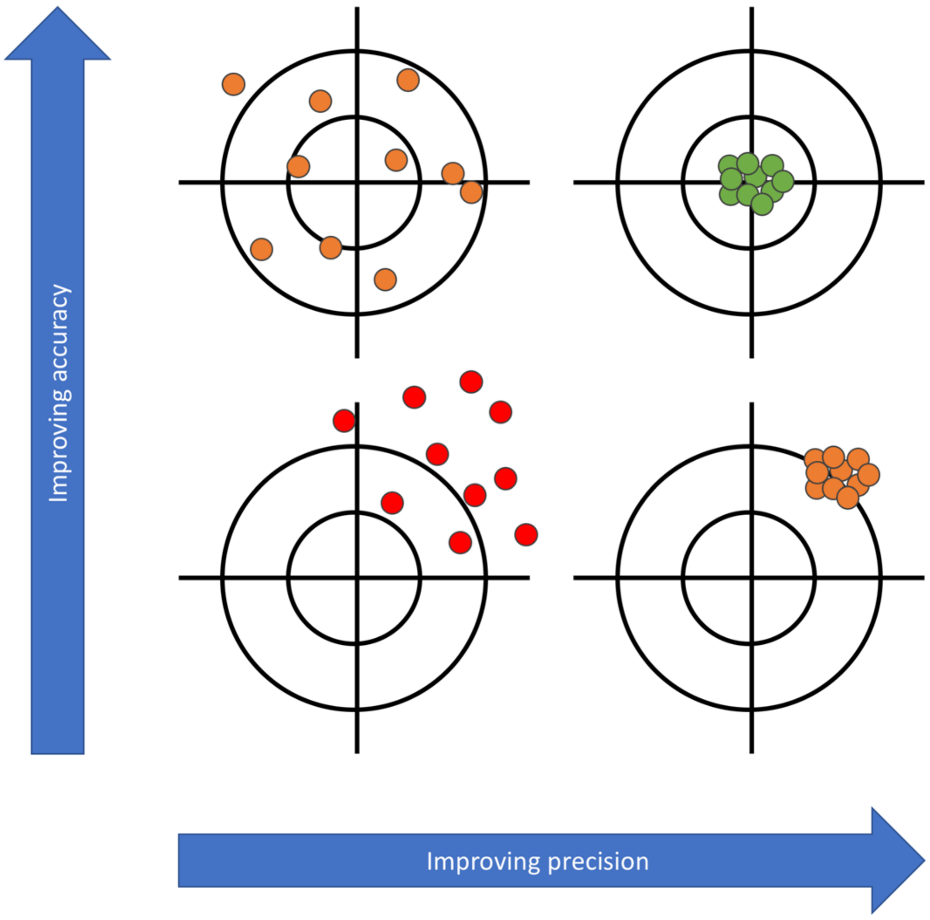

*Figure 43. Schematic depiction of accuracy and precision. Precision is improved by decreasing random errors, while accuracy is improved by decreasing systematic errors.*

In classification problems however, such as virtual screening or QSAR models, **accuracy** corresponds to the number of correct predictions (true positives and true negatives) made by the model over all kinds predictions made:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Accuracy is a good measure when the target variable classes in the data are nearly balanced, for example a compound database with 60% real actives and 40% real inactives. Given that high accuracies are indicative of models with a high TPR and a low FPR (see Figure 41), accuracy is a good metric to use in virtual screening experiments in which only limited budget is available to acquire novel compounds.

**Precision** in classification problems is a measure that tells what proportion of the predicted actives are real actives. The predicted actives that are real actives equals to *TP*, and the total number of actives in the database is *TP* + *FN*. Therefore, precision can be defined as:

$$Precision = \frac{TP}{TP + FP}$$

The sensitivity or recall gives information about a model's performance with respect to the false negatives (how many actives did we miss out), while precision gives information about its performance with respect to false positives (how many actives were retrieved from the database). Precision is about being precise: even if the database contained only one active compound, and if this active one was captured correctly, the method would be 100% precise. In contrast, recall is not so much about capturing cases correctly but more about capturing all actives. So basically, if we want to focus more on minimizing false negatives, recall should be as close to 100% as possible without precision being too bad. If one wants to minimize false positives, then focus should be to make precision as close to 100% as possible.

### *F1-score*

The F1 -core is a single score that represents both precision and sensitivity in a single number:

$$F1\ score = \frac{2 * precision * recall}{precision + recall}$$

## 3.2. Validation of continuous models

In order to be able to use the different performance metrics (see the previous section), continuous data need to be converted into a set of classifications. This is done by ranking the set from ‘good’ to ‘bad’ (or the reverse), and then applying a cutoff to divide the set into ‘good’ and ‘bad’ data (or ‘active’ versus ‘inactive’ compounds) (Figure 44).

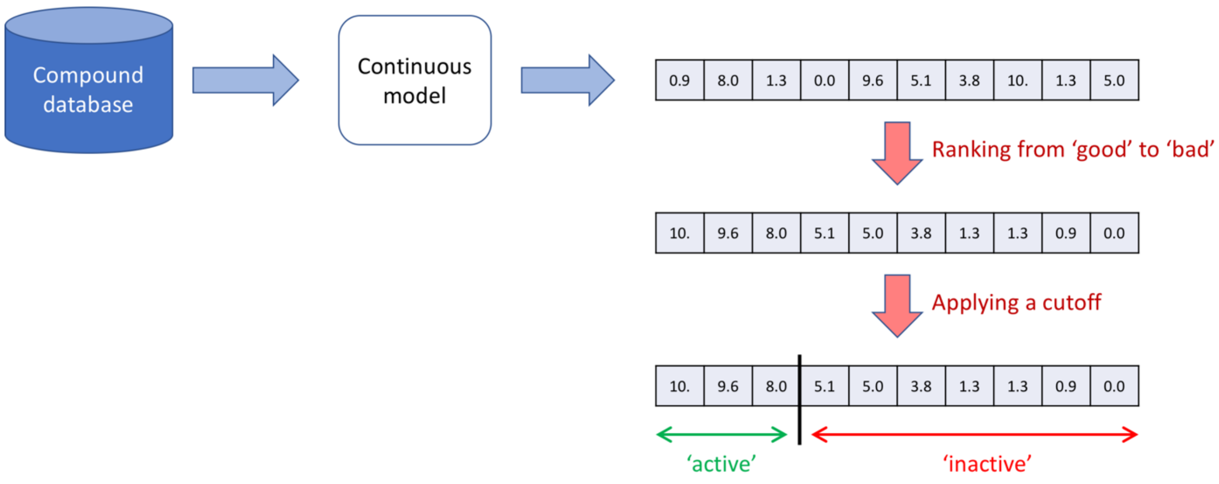

*Figure 44. Transforming continuous data into binary data. The first step is the ranking of the data according some user-defined criterium, and the second step is the categorization of the data into two classes by applying a cutoff. This cutoff value is user-defined and may affect the outcome of the performance metrics.*

The actual cutoff value has influence on the performance metrics such as specificity, accuracy, and all others, and should therefore be treated as part of the model parameters. As an example, consider two models (a bad and a good model) with their predicted activity of 10 compounds (of which the real experimental activities are known) and using 8 different cutoff values (Figure 45). The corresponding performance metrics are given in Table 9. From these it can be seen that the derived performance metrics are depending on the actual cutoff value, hence each time a performance metric is reported then the corresponding cutoff value should be given as well.

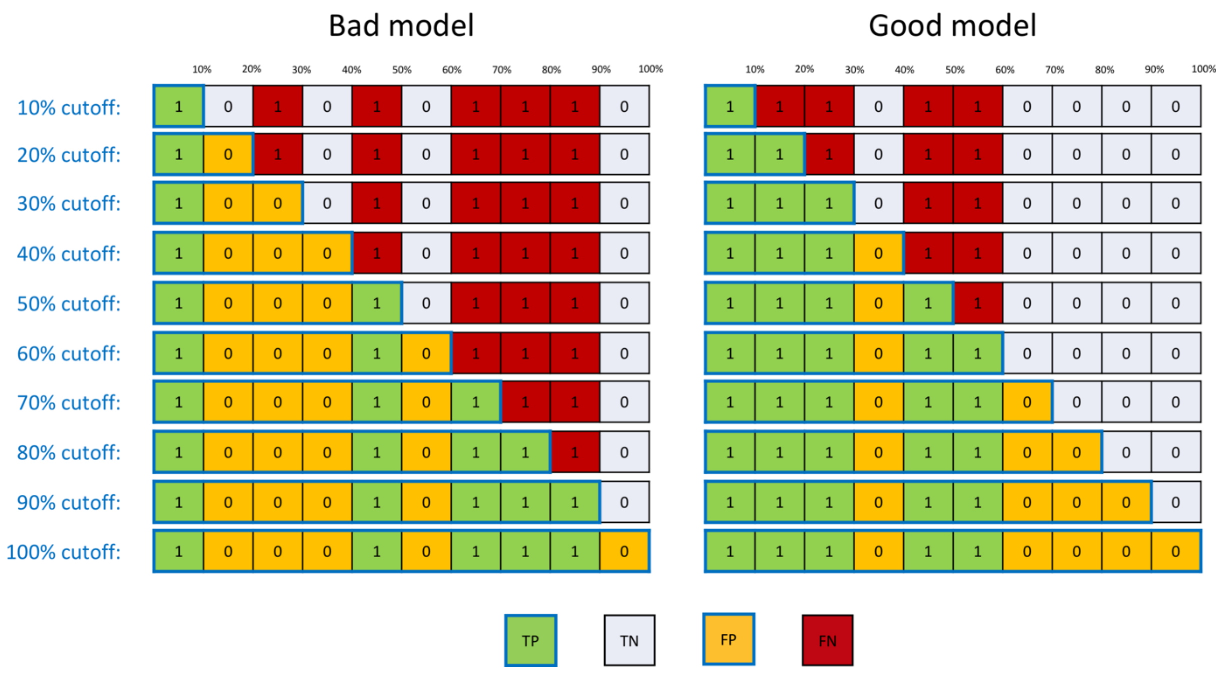

*Figure 45. True and false positives, and true and false negatives for two kinds of models. Left the bad model, right the good model. The cutoff ratio is also given. The '1' or '0' in each cell represents the real value of the cell ('1' indicates 'active', '0' is 'inactive'). All values within the defined cutoff are predicted 'active' by the model.*

*Table 9. Calculated performance metrics for the bad and good model shown in Figure 45.*

| Cutoff | TP | TN | FP | FN | ACC | PRE | SPE | TPR | FPR | F1 | EF | AUC |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | *Bad model* | | | | | | |
| *10%* | 1 | 4 | 1 | 5 | 0.45 | 0.50 | 0.80 | 0.17 | 0.20 | 0.25 | 0.92 | 0.48 |
| *20%* | 1 | 3 | 2 | 5 | 0.36 | 0.33 | 0.60 | 0.17 | 0.40 | 0.22 | 0.61 | 0.38 |
| *30%* | 1 | 3 | 3 | 4 | 0.36 | 0.25 | 0.50 | 0.20 | 0.50 | 0.22 | 0.55 | 0.35 |
| *40%* | 1 | 2 | 3 | 4 | 0.30 | 0.25 | 0.40 | 0.20 | 0.60 | 0.22 | 0.50 | 0.30 |
| *50%* | 2 | 2 | 4 | 3 | 0.36 | 0.33 | 0.33 | 0.40 | 0.67 | 0.36 | 0.73 | 0.37 |
| *60%* | 2 | 1 | 4 | 3 | 0.30 | 0.33 | 0.20 | 0.40 | 0.80 | 0.36 | 0.67 | 0.30 |
| *70%* | 3 | 1 | 4 | 2 | 0.40 | 0.43 | 0.20 | 0.60 | 0.80 | 0.50 | 0.86 | 0.40 |
| *80%* | 4 | 1 | 4 | 1 | 0.50 | 0.50 | 0.20 | 0.80 | 0.80 | 0.62 | 1.00 | 0.50 |
| *90%* | 5 | 1 | 4 | 0 | 0.60 | 0.56 | 0.20 | 1.00 | 0.80 | 0.71 | 1.11 | 0.60 |
| *100%* | 5 | 1 | 5 | 0 | 0.55 | 0.50 | 0.17 | 1.00 | 0.83 | 0.67 | 1.10 | 0.58 |
| | | | | | | *Good model* | | | | | | |
| *10%* | 1 | 5 | 0 | 4 | 0.60 | 1.00 | 1.00 | 0.20 | 0.00 | 0.33 | 2.00 | 0.60 |
| *20%* | 2 | 5 | 0 | 3 | 0.70 | 1.00 | 1.00 | 0.40 | 0.00 | 0.57 | 2.00 | 0.70 |
| *30%* | 3 | 5 | 0 | 2 | 0.80 | 1.00 | 1.00 | 0.60 | 0.00 | 0.75 | 2.00 | 0.80 |
| *40%* | 3 | 4 | 1 | 2 | 0.70 | 0.75 | 0.80 | 0.60 | 0.20 | 0.67 | 1.50 | 0.70 |
| *50%* | 4 | 4 | 1 | 1 | 0.80 | 0.80 | 0.80 | 0.80 | 0.20 | 0.80 | 1.60 | 0.80 |
| *60%* | 5 | 4 | 1 | 0 | 0.90 | 0.83 | 0.80 | 1.00 | 0.20 | 0.91 | 1.67 | 0.90 |
| *70%* | 5 | 3 | 2 | 0 | 0.80 | 0.71 | 0.60 | 1.00 | 0.40 | 0.83 | 1.43 | 0.80 |
| *80%* | 5 | 2 | 3 | 0 | 0.70 | 0.63 | 0.40 | 1.00 | 0.60 | 0.77 | 1.25 | 0.70 |
| *90%* | 5 | 1 | 4 | 0 | 0.60 | 0.56 | 0.20 | 1.00 | 0.80 | 0.71 | 1.11 | 0.60 |
| *100%* | 5 | 0 | 5 | 0 | 0.50 | 0.50 | 0.00 | 1.00 | 1.00 | 0.67 | 1.00 | 0.50 |

## *Enrichment factor (EF)*

The enrichment factor EF measures by how much the model is able to 'enrich' the number of actives in the predicted set of actives when compared to how many actives there exist in the entire dataset:

$$EF = \frac{TP(TP + TN + FN + FP)}{(TP + FP)(TP + FN)}$$

## *Mean squared error (MSE)*

The mean squared error is a simple metric that describes how well the predicted values match with the true values. It is calculated by taking the mean of all squared differences between the true values and the corresponding predictions:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

with n the number of datapoints in the test set, $\widehat{Y}_\iota$ the predicted value of datapoint *i*, and $Y_i$ the real value of datapoint *i*. In other words, the *MSE* is the mean of the squares of the differences between real values and predictions. If this difference is close to 0, then the *MSE* will also be close to 0, indicative of a good predictive model.

## *AUC-ROC*

AUC-ROC stands for Area Under the Curve – Receiver Operating Characteristics. It is one of the most important evaluation metrics for checking any classification model's performance. It is also sometimes written as AUROC (Area Under the Receiver Operating Characteristics).

AUC-ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between actives and inactives. Higher the AUC, better the model is at predicting actives as actives and inactives as inactives. By analogy, the higher the AUC, the better the model is at distinguishing between inactive and active compounds.

The ROC curve is plotted with the true positive rate (TPR) against the false positive rate (FPR), where TPR is on *y*-axis and FPR is on the *x*-axis (Figure 46):

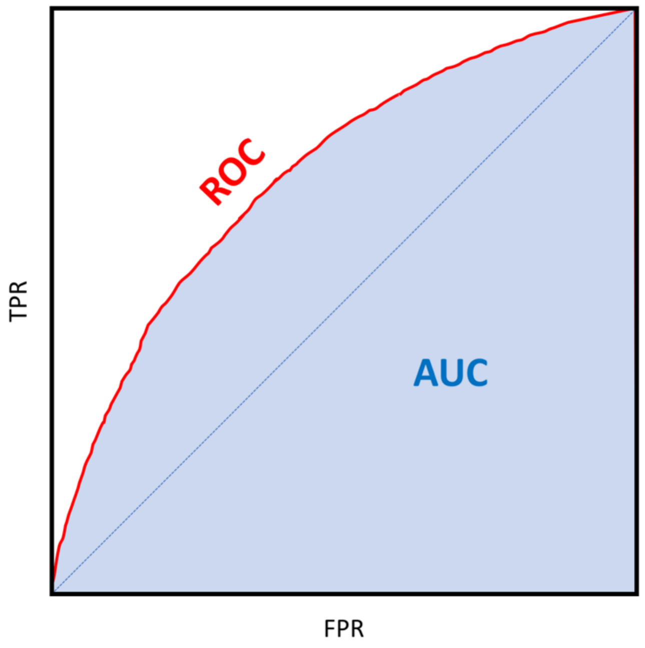

*Figure 46. Illustration of the AUC-ROC curve. The dotted blue line corresponds to an AUC = 0.5, and the red line has an AUC larger than 0.5.*

An excellent model has an AUC near to one which means it has good measure of separability. A poor model has AUC near to zero which means it has worst measure of separability. In fact, a model with AUC = 0 is reciprocating the result as it is predicting real actives as inactives and real inactives as actives. And when AUC is 0.5, it means that the model has no class separation capacity whatsoever.

Calculating the AUC from a single [FPR,TPR] pair of values obtained from the confusion matrix requires some area calculations. It is the sum of three areas as shown in Figure 47.

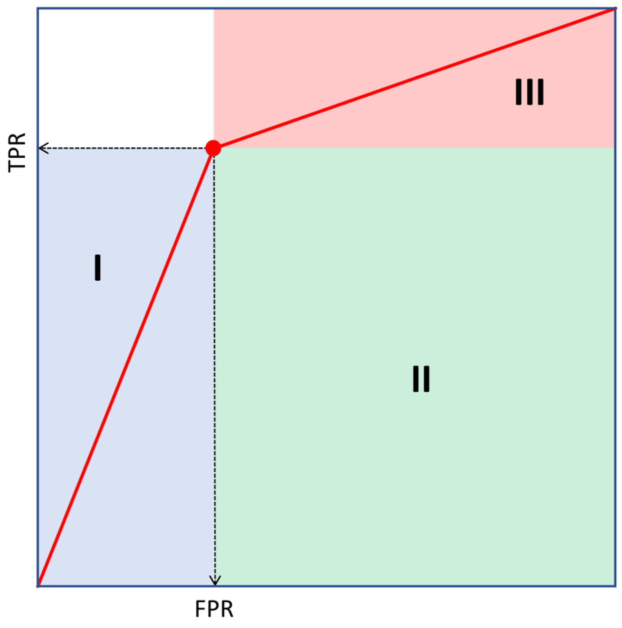

*Figure 47. Calculating the AUC-ROC value when only a single [TPR,FPR] pair of values is given. It becomes the sum of half the area of I, the entire area of II, and half the area of III.*

Mathematically:

$$AUC = \frac{TPR \times FPR}{2} + (1 - FPR) \times TPR + \frac{(1 - FPR) \times (1 - TPR)}{2}$$

which becomes:

$$AUC = \frac{TPR - FPR + 1}{2}$$

## 3.3. Cross-validation

Cross-validation or out-of-sample testing is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (called the validation dataset or testing set). The goal of cross-validation is to test the model’s ability to predict new data that were not used in estimating it, in order to flag problems like overfitting and to give an insight on how the model will generalize to an independent dataset.

A particular widely-used method of cross-validation is *k*-fold cross-validation. With *k*-fold cross-validation the complete dataset is divided into *k* disjoint parts of the same size. Subsequently *k* different models are built on *k*-1 parts each while those models are always validated on the remaining part of data. The picture below shows how a 10-fold cross-validation works (Figure 48):

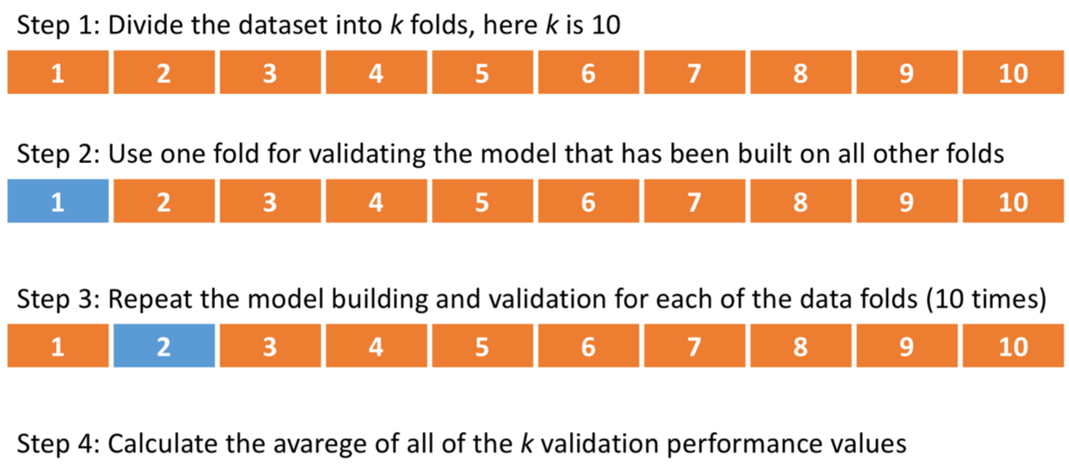
Step 1: Divide the dataset into *k* folds, here *k* is 10
1
2
3
4
5
6
7
8
9
10
Step 2: Use one fold for validating the model that has been built on all other folds
1
2
3
4
5
6
7
8
9
10
Step 3: Repeat the model building and validation for each of the data folds (10 times)
1
2
3
4
5
6
7
8
9
10
Step 4: Calculate the avarege of all of the *k* validation performance values

*Figure 48. Principle of a k-fold cross-validation. Divide the data into k disjoint parts and use each part exactly once for testing a model built on the remaining parts.*

For the reasons discussed above, a *k*-fold cross-validation is recommended whenever you want to validate the future accuracy of a predictive model. It is a simple method which guarantees that there is no overlap between the training and test sets. It also guarantees that there is no overlap between the *k* test sets which is good since it does not introduce any form of negative selection bias. And last but not least, the fact that you get multiple validation errors for different test sets allows you to build an average and standard deviation for these test errors. This means that instead of getting a test error like 15% you will end up with an error average like 14.5% +/- 2% giving you a better idea about the range the actual model accuracy will likely be when you put into production.