UNIVERSITA' DEGLI STUDI DI CAMERINO

School of Advanced Studies

DOCTORATE COURSE IN

"Physics"

XXXIV Cycle

TITLE OF THE THESIS:

"Ligand-Protein Binding Affinity Prediction Using Machine Learning Scoring Functions"

PhD Student

Ing. Francesco Pellicani

Supervisor

Prof. Sebastiano Pilati

Co-supervisor

Prof. Diego Dal Ben

Index

Introduction

Subject introduction

In recent years, artificial intelligence makes its appearance in extremely different fields with promising results able to produce enormous steps forward in some circumstances. In chemoinformatics the use of machine learning technique, in particular, allows the scientific community to build apparently accurate scoring functions for computational docking. These types of scoring functions can overperform classic ones, the type of scoring functions used until now. However the comparison between classic and machine learning scoring functions are based on particular tests which can favour these latter, as highlighted by some studies. In particular the machine learning scoring functions, per definition, must be trained on some data, passing to the model the instances chosen to describe the complexes and the relative ligand-protein affinity. In these conditions the scoring power of the machine learning scoring functions can be evaluated on different dataset and the scoring functions performance recorded can be different depending on it. In particular, datasets very similar to the one used for the training phase of the machine learning scoring function can facilitate in reaching high performance in the scoring power.

Project context

Due to their importance in drug discovery, the development of scoring functions have been the focus of intense research endeavors for decades. They are fundamental instruments for the first phase of new medicine or vaccine discovery. Normally the period necessary to produce a new product of this type vary between 10 and 12 years. The first phase is called drug discovery. It consists in a massive screening of small molecules called ligand in order to discover what are the active ones. In principle the affinity between a ligand and a protein can be measured experimentally in silico. Actually, because of the large quantity of ligands considered, the screening is performed using scoring functions. However the actual scoring functions are moderately reliable and a further experimental test phase is necessary to accurate rank the active ligands. For this reason the drug discovery phase last from 2 to 3 years which is about

the 20% of the entire period necessary to produce a new medicine or vaccine. A reliable scoring function can drastically reduce the period necessary for drug discovery because the experimental activity is not necessary.

In this period it is very easy to understand how important is reducing the time of medicine or vaccine production. In fact, in an epidemic context, as the one we are passing trough, the reduction of the time necessary to produce the COVID-Sars2 vaccine would have reduced the consequences of the Covid epidemic, first of all, the victims.

The problem of knowing the ligand-protein status and consequently deriving the relative affinity is extremely hard. In fact a close solution for the Schrödinger equations in most cases is impossible. Finding a way to have a reliable estimate of ligand-protein affinity is a challenging objective for the scientific community. Many studies are conducted on this subject since the classic scoring functions appeared. Despite a very deep interesting in the subject, both from industries, but also from authorities, in order to prevent risks due to epidemic, the progress in the accuracy of classic scoring functions is still not sufficient.

In recent years the incoming of artificial intelligence algorithms opens a new road to create more reliable scoring functions. The interest on the subject has just started and very few researchers are conducted on the topic. This work is one of these. Since the work on the present study started, many other studies on this subject have started or have been completed, attesting the increasing interest the topic. On the other hand, the subject is far from being completely explored. In fact, artificial intelligence has a fundamental element, necessary to be used to correctly work, a big database for training the model. Shared databases was born decades ago, anyway, in recent years, they are undergoing an intense development right for the incoming of artificial intelligence in chemioinformatic. They are cause and consequence at the same time in the increasing of artificial intelligence application. For this reason the potentiality of artificial intelligence in the field is still unexplored and lot of work on machine learning scoring function, but also in chemioinformatic in general, is necessary to be done. Right for this reason each new study on the topic counts on more reliable and developed database and can produce better and better results

Objectives

The objective of the present study is to verify the real efficiency and the effective performances of the new born machine learning scoring functions. Our aim is to give an answer to the scientific community about the doubts on the fact that the machine learning scoring function can be or not the revolutionary road to be followed in the field of chemioinformatic and drug discovery. In order to do this many tests are conducted and a definitive test protocol to be executed to exhaustive validate a new machine learning scoring function is proposed .

Here we investigate what are the circumstances in which a machine learning scoring function produces overestimated performances and why it can happen. As a possible solution we propose a tests protocol to be followed in order to guarantee a real performance descriptions of machine learning scoring functions. Eventually an effective and innovative solution in the field of machine learning scoring functions is proposed. It consists in the use of per-target scoring functions which are machine learning scoring functions created using complexes coming from a single protein and able to predict the affinity of complexes which use that target. The data used to build the model are synthetic and for this reason are easy to be created. The performances on the target chosen are better than the ones obtained with basic model of scoring functions and machine learning scoring functions trained on database composed by more than one protein.

Thesis outline

The present work is composed of 7 chapters. The following topics are treated.

Chapter 1

In Chapter 1 an overview on artificial intelligence with particular reference to machine learning and to the algorithms used in this study, is done.

Chapter 2

In Chapter 2 an overview on the subject selected to apply machine learning algorithms is done. In particular proteins, ligands, relative complexes and, ligand-protein affinity are

described. Once the subject are introduced, the database used in the study are presented. In particular experimental and synthetic data are presented with all their properties.

Chapter 3

In Chapter 3 a detailed presentation of the studies conducted on applications of artificial intelligence to chemioinformatic, with particular reference to machine learning scoring function, is presented. In particular an excursus on more important studies is done, with reference to the regression model used, the database used, and the results obtained.

Chapter 4

In Chapter 4 the scoring function built in this study, MLP scoring function, is presented. Initially the descriptive model used to represent the ligand-protein complex is presented and the relative code is described step by step. Then the regression model used for the MLP scoring function is detailed presented with particular attention to the training method and the network structure. The code used is described step by step.

Chapter 5

In Chapter 5 the MLP scoring function is tested on different types of test and the performances obtained are discussed and compared. The tests performed are the horizontal test, the vertical test and the per-target vertical test.

Chapter 6

In Chapter 6, considering the analysis made in the previous chapters, a possible solution, effective in the field of machine learning scoring function, is proposed. The solution is the per-target scoring function. The per-target scoring function is compared to other machine learning and other types of scoring functions in different tests.

Chapter 7

In Chapter 7 the conclusion of this study are taken in terms of: confirmation of the presence of bias conditions in a horizontal test; fairy test for performances evaluation of machine learning scoring functions with respect to classic scoring functions; innovative and effective solution in the field of scoring function.

Summary of research work and main findings

In order to pursue the objective specified, as preliminary operations, we create an appropriate experimental and synthetic database; we design and test different training protocols, network structure and complex descriptive model and verify the best; we choose the most suitable types of test to verify the performances of the scoring functions.

The study confirms that machine learning scoring functions perform excellently in horizontal test. This test describes a particular utilizing case of a scoring function and overestimate their general performance. The typical use of a scoring function in the field of chemistry and pharmacy emerges to be described by a vertical test. In these conditions the MLP scoring function, as other machine learning scoring functions, shows such a degradation of performances that they become similar to the ones of classic scoring functions. For this reason we propose that performance of new machine learning scoring function should always be described by both horizontal and vertical tests.

Eventually we introduce machine learning per-target scoring function. It guarantees always higher performance with respect to the one of machine learning scoring function in vertical test performed on the same test set. In addition it seems to have improvable performance if larger training database would be used.

1 Machine Learning

In this chapter an overview on artificial intelligence is done and the theory on which the study is based is presented.

1.1 Artificial intelligence and machine learning

1.1.1 Artificial Intelligence

Machine learning is a branch of artificial intelligence (AI) (1) (2) (3). Artificial intelligence is a discipline which deals with the creation of machines able to imitate the capabilities of human intelligence using different types of algorithms (4) (5). Going into details of artificial intelligence definition, it can be classified as the discipline which develops algorithms that allow the machines to produce intelligent activities at least in some specific domains. Starting from this definition, and considering that the exact functioning of the human brain is still partially unknown, it is clear that artificial intelligence is a very vast and faceted research area. Nowadays this field of research is highly developed because of the technological level reached in the computational calculation (6). The hardware systems are very powerful and the size is reduced. The energetic losses are low. In addition the capability of analyzing large databases of any type of data in very short times encourages the development of artificial intelligence research.

The scientific community's interest for artificial intelligence starts decades ago (7). The first project of artificial intelligence started in 1943 when two researchers, Warren McCulloch and Walter Pitt, proposed to the scientific community the first artificial neuron (8). In 1949 Donald Olding Hebb, a Canadian psychology, published a book in which the connections between artificial neurons and the complex model of real brains are analyzed in detail.

The first prototypes of functioning artificial neural networks appeared in the ‘50. These prototypes are mathematical or informatics models developed to reproduce the real functioning of human neurons and to solve problems in a similar way to a human mind. The public interest increased thanks to the young Alan Turing, who still in 1950 tried to explain how a computer can perform as a human brain.

The term “artificial intelligence” officially appeared for the first time thanks to the mathematician John McCarthy in 1956. In this context the first programming languages specific for artificial intelligence were launched in 1958 (Lisp) (9) and in 1973 (Prolog) (10). From that moment the story of the artificial intelligence was swinging. It was characterized by significant steps forward from the point of view of the physical and mathematical models which became more and more complex in order to imitate some human brain functionality. On the other hand, there was a sort of lack in the hardware and neural network research.

The first model of artificial neural network appeared at the end of the ‘50. It was called the perceptron. It was proposed in 1958 by the psychologist Frank Rosenblatt (11). It was a network with an input layer and an output layer and a learning rule based on the error back-propagation algorithm. The mathematical function modifies the weights of the connections causing a difference between the effective output and the desired output. Some experts identify in the Rosenblatt’s perceptron birth of the artificial intelligence.

In the following years the mathematicians Marvin Minsky and Seymour Papert demonstrated the limits of the neural network proposed by Rosenblatt (12). In fact the perceptron was able to recognize, after being trained, only linear separable functions. In addition the calculation capability of a single perceptron was limited and the performances were strongly dependent on the choices of the inputs and of the algorithms used to modify the connection weights and, consequently, the outputs. The two mathematicians Minsky and Papert guessed that more than a single layer of perceptron in the artificial neural network can solve more complex problems. However, in those years, the increasing computational resources required in the training of more complex networks did not find an answer in the hardware development.

The first important turning point in hardware development arrived in the ‘90 with the introduction of the graphics processing unit (GPU) in the global market. The GPU is faster than the old central processing unit (CPU) and is able to support complex processes. They operate at lower frequency and consume less energy with respect to CPU (13).

In the last decade another important step forward was done with the development of neuromorphic chip. This chip integrates data elaboration and data storage in a single micro component in order to emulate the sensory and cognitive functions of the human

brain (14). This evolution was possible thanks to the acceleration of the research in the field of nanotechnology.

Through this global evolving process, the artificial intelligence is living an extremely strong development in the very recent years and it is applied in almost every sectors if it is possible and useful.

Artificial intelligence can be subdivided in many areas according to the functionality considered. The mean functionalities are understanding, reasoning, interacting and learning. The understanding is the capability to recognize texts, images, voices,...

Examples of artificial intelligence applications are search engines (Google Search,...), recommendation systems (offered by Netflix, Zalando, Amazon), driving internet traffic, targeted advertising (Instagram, Facebook), virtual assistants (Cortana, Alexa), autonomous vehicles (including drones and self-driving cars), automatic language translation (Microsoft Translator, Google Translate), facial recognition (Apple's Face ID, Microsoft's DeepFace), image labeling (used by Facebook, Apple's iPhoto, TikTok) and spam filtering.

There are also thousands of successful artificial intelligence applications used to solve problems for specific industries or institutions. A few examples are energy storage, medical diagnosis, military logistics or supply chain management. Also game playing counts numerous artificial intelligence applications.

The reasoning implies the capability of making logical deduction linking the collected information. The interaction is the capability of relating with the external environment. The learning is the capability of analyzing inputs and consequently producing outputs. Machine learning (ML) is a fundamental concept of artificial intelligence research since the interest on the subject began.

1.1.2 Machine learning

The first time the term “machine learning” was used was in 1959 by the scientist Arthur Lee Samuel (15). He defines the machine learning as the:

“field of study that gives computers the ability to “learn” (e.g., progressively improve performance on a specific task) with data, without being explicitly programmed.”

Today the definition most often adopted in the scientific community is the one proposed by the American scientist Tom Michael Mitchell, director of the Machine Learning department of Carnegie Mellon University (16):

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In other words, machine learning allows the machine to learn from the experience. It means that the particular performances of the machine improve after completing a task, both if rightly or wrongly.

The machine learning algorithms use computational methods to learn information directly from the data without using mathematical models or pre-determined equations. The machine operates without a programming code which indicates what to do step by step and for any occurrences. It uses only a set of data which are processed using specified algorithms. Using this method, the machine develops a proper logic to perform its task. The machine learning algorithms increase their performance in an adaptive way, with the increase of the data in the database.

According to Arthur Samuel at the end of 1950, machine learning uses 2 types of approaches: unsupervised machine learning and supervised machine learning. They differ in the learning type. Actually there are some subsets of machine learning types which allow a more detailed classification of machine learning categories (17) (18). In particular we consider 3 types of machine learning:

- Unsupervised machine learning (19)
- Supervised machine learning (20)
- Reinforcement learning (21)

Reinforcement learning is concerned with how software agents ought to take actions in an environment in a manner to maximize some notion of cumulative reward. In machine learning, the environment is usually represented as a Markov decision process (MDP) (22). Reinforcement learning algorithms are used when exact models are impraticable because they do not assume knowledge of a precise MDP mathematical model. Reinforcement learning algorithms are used in the game environment, when the aim is to learn to play a game against an intelligent subject, or in autonomous vehicles.

In unsupervised machine learning we have data without labels/target-values.
The instances of the data are described by D-dimensional vectors.

$$\{\boldsymbol{x}_i\}_{i=1}^{N}$$

The components of the vector, x_i, are the features which describe each datum. There is a vector for each of the N instances in the database
The aim of this method is to find a recurrent behaviour or a structure or some clusters in the data. It is mostly used for applications like the ones described in the following.

- Principal Component Analysis (PCA) for dimensionality reduction (23).

This method is used for changing higher-dimensional data to a smaller space. For example it is used to switch from 3D to 2D. The objective is to have a smaller dimension of data (2D instead of 3D), while the information, inside them, is preserved.

- Clustering (24).

Cluster analysis is the subdivision of a set of data into subsets (called clusters). Data within the same cluster are similar according to one or more pre-designed criteria, while data from different clusters are dissimilar. Different clustering techniques make different assumptions on the structure of the data and so can produce different subdivisions.
The clusterization is defined by some similarity metric. Then, it can be evaluated by internal compactness or by the similarity between members of the same cluster or by the differences between members of different clusters. One of the principal methods of clustering is K-means. It creates a partition in which each datum belongs to the cluster with the nearest mean (cluster center). The cluster center is considered the prototype of the cluster. K-means clustering minimizes variances within the cluster (squared Euclidean distances).

- Random forest (25).

Random Forest operates by constructing a multitude of decision trees at training time. Decision tree learning is one of the predictive modeling approaches used in machine learning. It uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item target value (represented in the leaves). A tree is built by splitting the source set, constituting the root node of the tree into subsets, which constitute the successor children. The splitting

is based on a set of splitting rules based on classification features. This process is repeated on each derived subset in a recursive manner called recursive partitioning. Random Forest can be used as unsupervised machine learning, but also as supervised machine learning. In this way it can be used as a dissimilarity measure in a set of data. Often unsupervised learning is used also for singular value decomposition, self-organized maps and a first step before supervised learning.

In supervised machine learning we have data that describe instances with a label/target-values.
The following expression indicates the features vectors and the relative labels.

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$$

The components of the $\boldsymbol{x}_i$ vectors are the features which describe each datum. The labels y_i are the target values of the data. Labels can have continuous values (regression) or discrete values (classification), e.g., 0 or 1 (Figure 1.1). N is the size of the dataset.

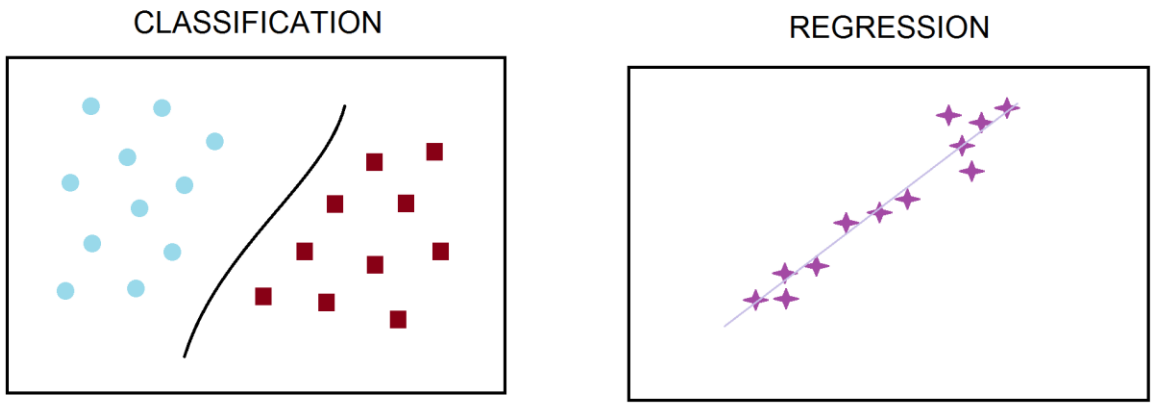

Figure 1.1 Label types

The aim of this method is to identify a model which links the data to the label/target-values. The model can reproduce those labels/target-values and can make new predictions of the label/target-value for new instances given their descriptive data.
It is mostly used for applications like the following:

- Regression analysis (linear regression, polynomial regression, logistic regression) (26).

Regression analysis encloses a large variety of statistical methods to estimate the relationship between features (input) and their associated labels (output). In the linear regression a line is used to best fit the given data according to a mathematical criterion which for example can be the least squares. The polynomial regression is used when a nonlinear problem is encountered. In this case a n degree polynomial is used to fit the given data. The logistic regression basically uses a logistic function to model a binary dependent variable.

- Support Vector Machines (SVM) (27).

The Support Vector Machines training algorithm in most cases is a non-probabilistic, binary, linear classifier. It builds a model able to classify the input data.

- Other applications of supervised machine learning are: k-nearest neighbours (algorithm used in pattern recognition for the classification of objects based on the characteristics of objects close to the one considered) and artificial neural networks which is the one used in this study.

1.2 Artificial neural network description

Artificial Neural Networks (ANNs) are computing systems inspired by the neural networks of human brains (28).

"*Any function can be approximated by a sufficiently deep artificial neural network*" K.-I. Funahashi, Neural Networks 2, 183-192 (1989)

Originally the artificial neural networks were created to solve problems in the same way that a human brain would. This system is trained on example data. The training allows the system to learn to operate a specific function without having been programmed specifically for this function. The artificial neural networks, as anticipated, are structurally similar to a human brain. In fact they are composed of nodes, called artificial neurons, which emulate the biological neurons. Each artificial neuron is connected with the others and the information can be transmitted from one neuron to another. Once the signal is received, each artificial neuron can process the signal before transmitting it to the next. In fact in the human brain the synapses transmit information among the neurons. The signal transmitted by each artificial neuron is a real number. The output of

each artificial neuron is a particular function of the inputs. The function used is called activation functions. They can be non-linear functions. Some weights are used to increase or decrease the strength of the signal at a connection. They are adjusted as learning proceeds. Typically, artificial neurons are aggregated into layers. Different layers may use different non-linear activation function on their inputs. Signals travel from the first layer (the input layer) to the last layer (the output layer) through one or more hidden layers. A neural network with a single hidden layer can approximate any continuous, multi-input/multioutput function with arbitrary accuracy (G. Cybenko, 1989; Kurt Hornik, 1991) However, the width of such networks might move to be exponentially large. Consider a general artificial neural network:

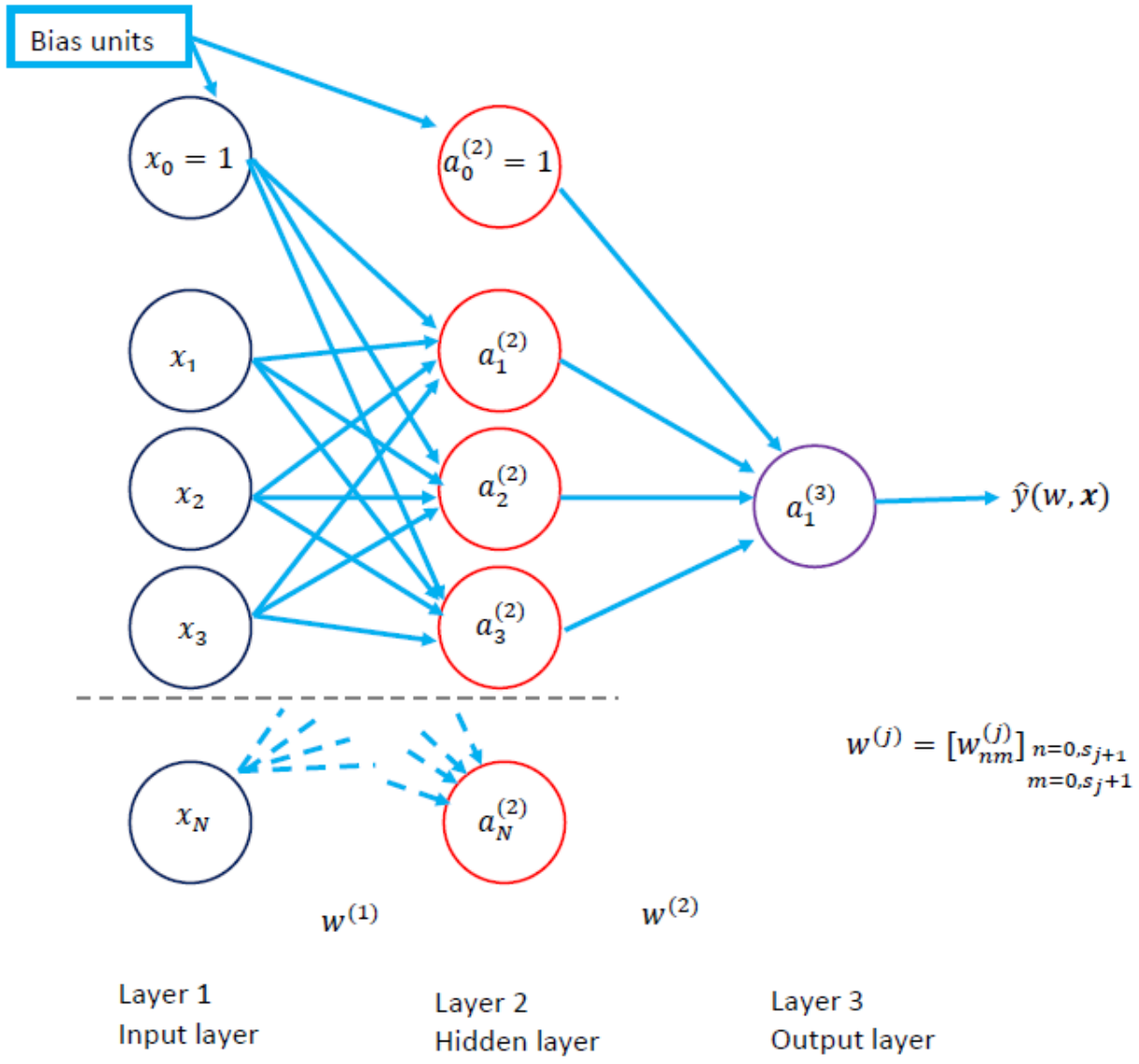

Figure 1.2 Generic artificial neural network composed of one hidden layer

The artificial neural network represented in Figure 1.2 is composed of one hidden layer plus the input and output layer. The network can be composed by any number of hidden layers, its behaviour is the same of the one described hereafter. The artificial neural network represented uses an input vector of 3 components. The number of neurons of the hidden layer is 3. Also the input vector components and the number of neurons can be varied, the relation among inputs and outputs are analogue.

$\boldsymbol{x}$ is the input vector. The instances x_i are represented inside the blue circles. They constitute the input layer. The neurons of the hidden layer are represented by the red circles. The output layer is composed of 1 neuron. It is represented by the violet circle.

$a_i^{(j)}$ is the activation function of neuron i in layer j.

The output vector is $\hat{\boldsymbol{y}}$. It depends on w and $\boldsymbol{x}$.

w is the matrix of weights. $w^{(j)}$ is the matrix of weights from layer j to layer $j+1$, $w^{(j)} \in R^{s_{j+1}\times(s_j+1)}$, where s_j is the number of units (without bias) in layer j. So $w^{(j)} = [w_{nm}^{(j)}]_{\substack{n=0,s_{j+1}\\ m=0,s_j+1}}$. $w_{nm}^{(j)}$ is the weight between neuron m at layer j and neuron n at layer $j+1$.

The terms with subscript 0 are the bias unit (x_0). In the case they are referred to the hidden layer, the reference layer is indicated in the apex ($a_0^{(2)}$) . The algorithmic bias is used to prevent systematic and repeatable errors that create unfair outcomes, such as privileging one arbitrary group of users over others. It is treated as an additional input to the artificial neural network. It is standard pre-set to 1.

In this neural network, information moves in one direction, forward with respect to entry nodes, through hidden nodes to exit nodes.

The outputs of layer 2 are:

$$a_1^{(2)} = h\left(w_{10}^{(1)}x_0 + w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3\right) = \left(w_1^{(1)}a^{(1)}\right)$$

$$a_2^{(2)} = h\left(w_{20}^{(1)}x_0 + w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3\right) = \left(w_2^{(1)}a^{(1)}\right)$$

$$a_3^{(2)} = h\left(w_{30}^{(1)}x_0 + w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3\right) = \left(w_3^{(1)}a^{(1)}\right)$$

Where $a^{(j)}$is the vector $(a_1^{(j)}, a_2^{(j)}, \dots, a_n^{(j)})$ and $a_i^{(j)}$, which is the activation function of neuron i in layer j, is given by the scalar product between $w_i^{(j-1)}a^{(j-1)}$, h is the function chosen as activation function.

This notation can be also substituted by the vectorized notation, which means evaluating $h(\boldsymbol{x})$ on each element of $\boldsymbol{x}$:

$$a^{(2)} = h(w^{(1)}a^{(1)})$$

The outputs of layer 3, that is the final output, is:

$$\hat{y}(w, \boldsymbol{x}) = a_1^3 = h^*\left(w_{10}^{(2)}a_0^{(2)} + w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)}\right) = h^*\left(w_1^{(2)}a^{(2)}\right) \Rightarrow \hat{y}(w, \boldsymbol{x}) = a^{(3)} = h^*\left(w^{(2)}a^{(2)}\right)$$

Where h^* is the activation function of the output unit (e.g, identity).
If we consider the simplest artificial neural network, reported hereafter (Figure 1.3), the following output is obtained. The network is composed of one neuron (the output neuron), represented by the orange circle. No hidden layers are present. The input vector $\boldsymbol{x}$ has 2 components: x_1, x_2. The output is $\hat{y}$.

$$\hat{y}(w, \boldsymbol{x}) = h(w_{10} + w_{11}x_1 + w_{12}x_2) = h\left(\sum_{i=0}^{2} w_{1i}x_i\right)$$

where h indicate the activation function.

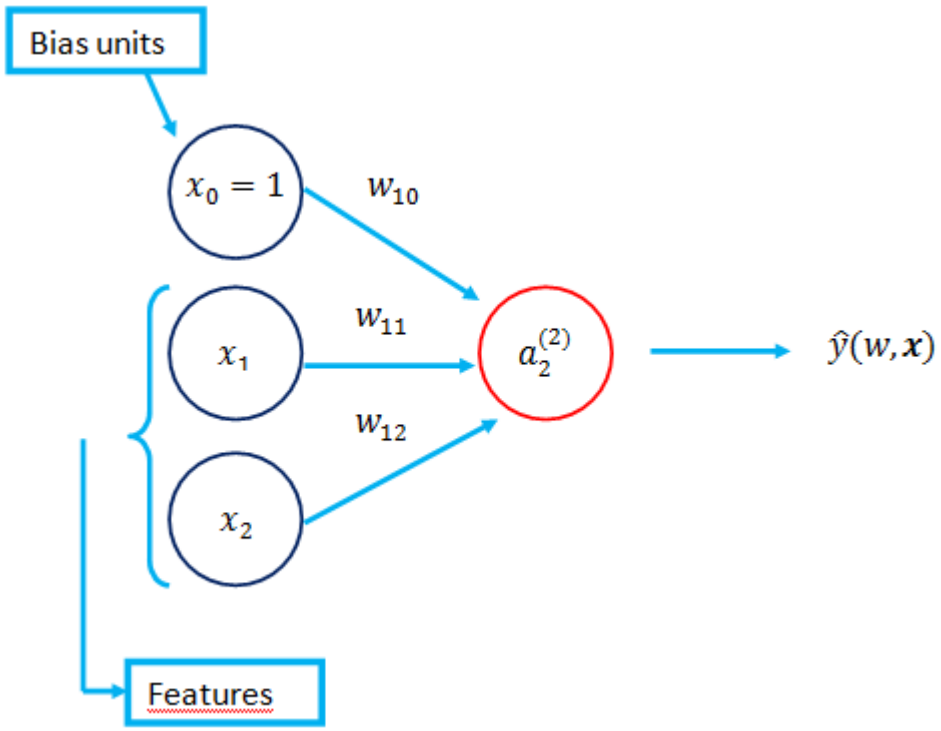

Figure 1.3 Simplest artificial neural network composed of a single neuron, the output neuron. Input vector composed of two components

In a neural network an activation function defines how the weighted sum of the input is transformed into an output from one or more nodes in a layer of the network. Activation functions are a key part of neural network design (29). The activation function choice has a large impact on the performance and capability of the neural network. Different activation functions may be used in different parts of the model. Usually artificial neural networks are designed to use the same activation function for all nodes in a layer and typically all hidden layers use the same activation function. The output layer can use a different activation function depending on the type of prediction problem. Many different types of activation functions can be used in neural networks, although only few functions are normally used in practice for hidden and output layers. Hereafter the most important activation functions are reported (Figure 1.4).

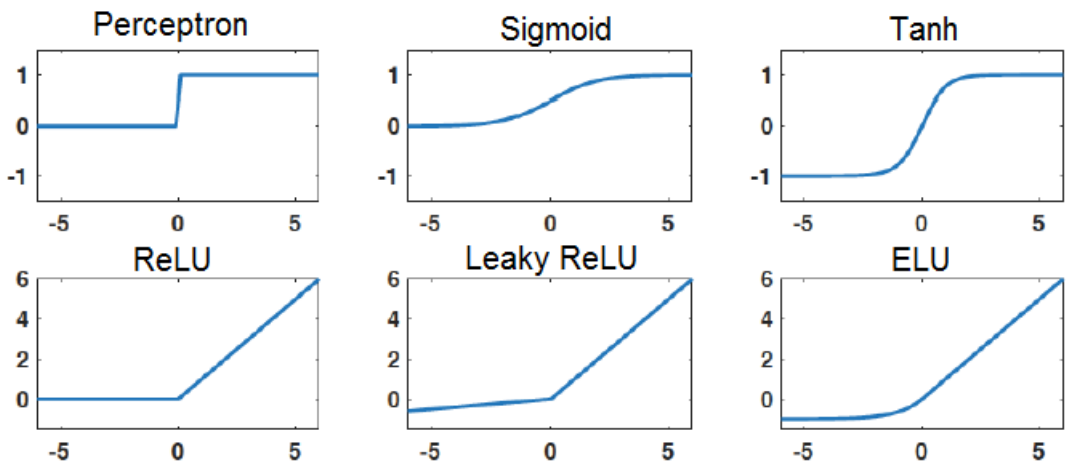

Figure 1.4 The most important activation functions

The perceptron or Heaviside step function is calculated as follows:

$$H(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}.$$

It is a step function, the value of which is zero for negative arguments and one for positive arguments. The value of the function in 0 varies depending on the function definition. Some common choice can be that $H(0) = 1, \frac{1}{2}, 0$. The function takes any real value as input.

The Sigmoid activation function is calculated as follows:

$$S(x) = \frac{1}{1 + e^{-x}}.$$

The function takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1, whereas the smaller the input (more negative), the closer the output will be to 0.

The Tanh (hyperbolic tangent) activation function is calculated as follows:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

It is very similar to the Sigmoid activation function. The function takes any real value as input and the outputs values are in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1, whereas the smaller the input (more negative), the closer the output will be to -1.

The rectified linear activation function, or ReLU activation function, is perhaps the most common function used for hidden layers.
The ReLU function is calculated as follows:

$$R(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases} = max\{0, x\}.$$

This means that if the input value (x) is negative, then a value 0 is returned, otherwise, the value is returned.

The leaky rectified linear unit (Leaky ReLU) is calculated as follows:

$$L(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases}.$$

The function is very similar to the ReLU activation function. The function takes any real value as input and outputs values are in the range $-\infty$ to $+\infty$. It is a growing function without asymptotes

The Exponential linear unit (ELU) is calculated as follows:

$$E(x) = \begin{cases} \alpha(e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$

The function is very similar to the ReLU activation function. The function takes any real value as input and outputs values are in the range - α to $+\infty$. It is a growing function starting from -1. In some case a parameter α is used as multiplier in front of the expression $(e^x - 1)$ only for negative or zero input.

1.3 Neural network training method

The process of supervised machine learning adopted in artificial neural networks consists in learning a function that can be used to predict the output associated with new inputs through iterative optimization of a loss function or cost function. The cost

function is a kind of function of the difference between estimated and true values for an instance of data. The optimization consists in optimizing the parameter of the model in order to minimize the cost function. The iteration permits to improve the accuracy of the function outputs or predictions over time. For this reason it is said to have learned to perform that task.

If we consider the simple example of a linear regression with data composed of one dimensional feature, the database is the following:

$$\{(x_1, y_1), \dots, (x_i, y_i), \dots, (x_n, y_n)\}$$

If a linear model is used, the output of the model is:

$$\hat{y}(\Theta, \boldsymbol{x}) = \theta_0 + \theta_1 x = X \cdot \Theta$$

Where $X = (1 \quad x)$ and $\Theta = \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix}$

Θ is the matrix of the parameters. In this case is a one-dimensional vector with two components.

The cost function considered in this case is the squared difference between the predicted data and the real data. This is called the least squares method.

$$L(\Theta) = \frac{1}{N} \sum_{i=1}^{N} (\hat{y}(\Theta, \boldsymbol{x}_i) - y_i)^2$$

The optimization of the model parameters in order to have the best fit between predicted and real value is obtained minimizing the cost function: $\hat{\Theta} = arg\min_{\Theta} L(\Theta)$

The minimization of the cost function is done by taking the derivative of the function with respect to the variables. This operation allows us to obtain the optimal values of the parameters of the function (the vector Θ).

$$\frac{\delta L}{\delta \theta_0} = 2 \sum_{i=1}^{N} (\theta_0 + \theta_1 x_i - y_i) = 0$$

$$\frac{\delta L}{\delta \theta_1} = 2 \sum_{i=1}^{N} (\theta_0 + \theta_1 x_i - y_i) x_i = 0$$

The system is composed of 2 equations and we have 2 unknowns.

$$\theta_0 N + \theta_1 \sum_{i=1}^{N} x_i = \sum_{i=1}^{N} y_i$$

$$\theta_0 \sum_{i=1}^{N} x_i + \theta_1 \sum_{i=1}^{N} x_i^2 = \sum_{i=1}^{N} x_i y_i$$

The solution of this system is the following:

$$\theta_1 = \frac{\sigma(x,y)}{\sigma^2(x)}$$

$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

where σ and σ^2 are respectively the covariance of (x,y) and the variance of (x) and $\bar{x}$ and $\bar{y}$ are the average values.

$$\sigma(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})$$

$$\sigma^2(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \,;\;\; \bar{y} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

1.3.1 Gradient descent

In a general case, each data is described by k features. Because of this the training data is represented by a matrix composed of N rows, where N is the number of data in the database, and k columns. The parameters vector is composed by k elements. The loss function is the following.

$$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(\Theta, \boldsymbol{x}_i) - y_i)^2$$

In this case the optimization of the parameters is done using the gradient descent (30). It is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The principle is to take repeated steps in the opposite direction of the gradient of the function at the current point, because this is the direction of steepest descent.

Consider a function $F(x)$ for which the minimum has to be found. One starts with a guess x for a local minimum of F and considers the sequence $x_0, x_1, x_2, \ldots$ such that

$$x_{n+1} = x_n - \eta \nabla F(x_n), \qquad n \geq 0$$

Here, η is a real parameter called learning rate. The result is a monotonic sequence

$$F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots$$

Hopefully, the sequence $\{x_n\}$ converges to the desired absolute minimum. The value of the step size indicated by η is the learning rate of the algorithm. A graphic explanation of the gradient descent can be found in Figure 1.5.

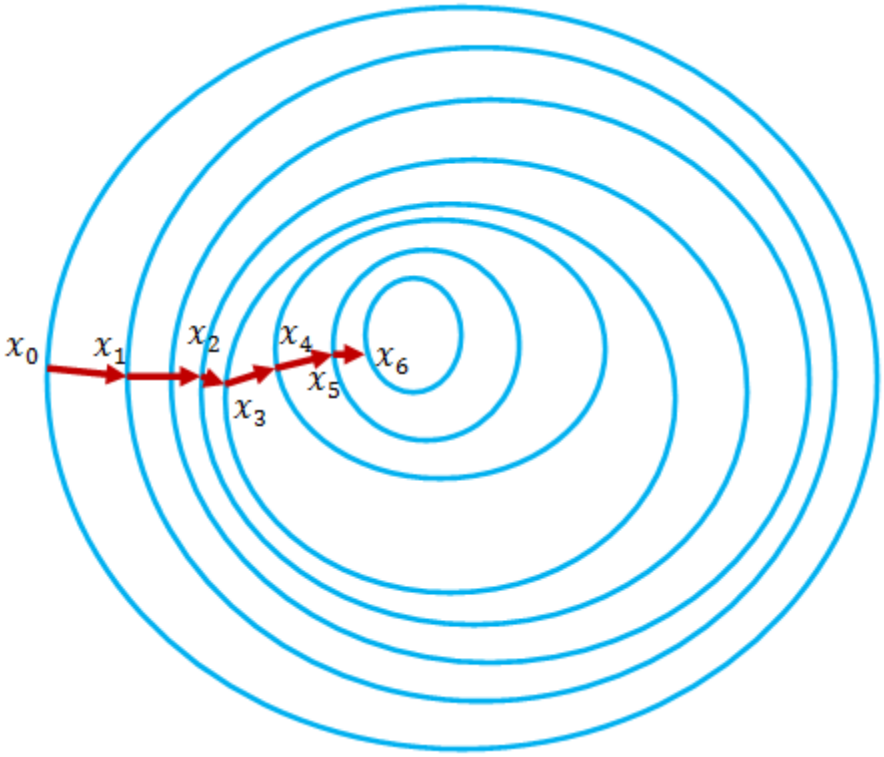

Figure 1.5 Graphic explanation of the gradient descent algorithm

When the gradient descent is applied to the loss function of linear regression, you obtain:

$$\frac{\delta L}{\delta \theta_0} = \frac{2}{N_t}\sum_{i=1}^{N_t}(X_i \cdot \Theta - y_i)$$

$$\frac{\delta L}{\delta \theta_j} = \frac{2}{N_t}\sum_{i=1}^{N_t}(X_i \cdot \Theta - y_i)x_{i,j} \quad \text{for } j = 1, \dots k$$

Where $x_{i,j}$ is the j^{th} element of the feature vector $\boldsymbol{x}_i$ that represent the i^{th} instance. In order to possibly achieve the global minimum, you start with a random value for θ_j for $j = 0, \dots, k$. Next value of θ_j, $\theta_j{}'$, is obtained with the following operation:

$$\theta_j{}' = \theta_j - \eta \frac{\delta}{\delta \theta_j} L(\Theta).$$

In the case the number of data are numerous (typically $N_t \approx 10^{4-5}$) the stochastic gradient descent can be used to avoid calculation too expensive in terms of resources. Instead of computing the gradient using all N_t instances of the training set, it consist in using mini-batches randomly chosen of 10 to 100 or more instances, depending on the database size, for which the gradient is computed. The computed gradient points on average in the right direction, with a stochastic noise that helps escaping local minima.

1.3.2 Gradients computation in artificial neural networks: backpropagation algorithm

In the case an artificial neural network, with many neurons and layers and a standard size database, is used, it is possible to have 10^4 or even 10^5 parameters, to be computed, for every instance of the mini-batch. In such cases the backpropagation algorithm is used for evaluating the gradient of the loss function (31).

The aim is to evaluate $\frac{\delta L(W)}{\delta w_{nm}^{(j)}}$ for any instance i, considering that the loss function, without regularization, is additive.

The weighted input of neuron n at layer j is:

$$z_n^{(j)} = \sum_{m=0}^{s_{j-1}} w_{nm}^{(j-1)} a_m^{(j-1)}$$

The relative activation is:

$a_n^{(j)} = h(z_n^{(j)})$ (vector notation: $a^{(j)} = h(z^{(j)})$.

It is possible to define the "error" of neuron n at layer j as:

$\delta_n^{(j)} = \frac{\delta L}{\delta z_n^{(j)}}$ (vector notation: $\delta^{(j)} = \begin{pmatrix} \delta_1^{(j)} \\ \delta_2^{(j)} \\ \vdots \\ \delta_{s_j}^{(j)} \end{pmatrix}$).

Instead, in the last layer, the error is:

$\delta_1^{(L)} = \frac{\delta L}{\delta a_1^{(L)}} h^{*\prime}(z_1^{(L)})$ (1 exit neuron); $\delta^L = \nabla_{\mathrm{a}^{(\mathrm{L})}} \mathrm{L} \otimes \mathrm{h}^{*\prime}(\mathrm{z}^{(\mathrm{L})})$ (many exit neurons).

In the latter case $\otimes$ indicates the Hadamard product (elementwise).

Starting from these assumptions, according to the backpropagation algorithm, the error at layer j, given error at layer $j+1$, is:

$\delta^{(j)} = ((w^{(j)})^T \delta^{(j+1)}) \otimes \mathrm{h}'(\mathrm{z}^{(\mathrm{j})})$ for $j = L-1, \ldots, 2$.

The partial derivatives with respect to weights and biases are respectively:

$$\frac{\delta L}{\delta w_{mn}^{(j)}} = a_n^{(j)} \delta_m^{j+1}; \frac{\delta L}{\delta b_m^{(j)}} = \delta_m^{j+1}$$

Layer 1 adds the term due to regularization: $\frac{\delta L}{\delta w_{mn}^{(j)}} \rightarrow \frac{\delta L}{\delta w_{mn}^{(j)}} + \lambda w_{mn}^{(j)}$

An alternative notation can be used:

$$\frac{\delta L}{\delta w} = a_{in} \delta_{out}$$

where a_{in} is the activation of the neuron input to w, δ_{out} is the error on the output neuron.

1.3.3 Overfitting and underfitting

The function learned by the artificial neural network can reproduce the data distribution more or less accurately (32). In particular the following situation can be encountered.

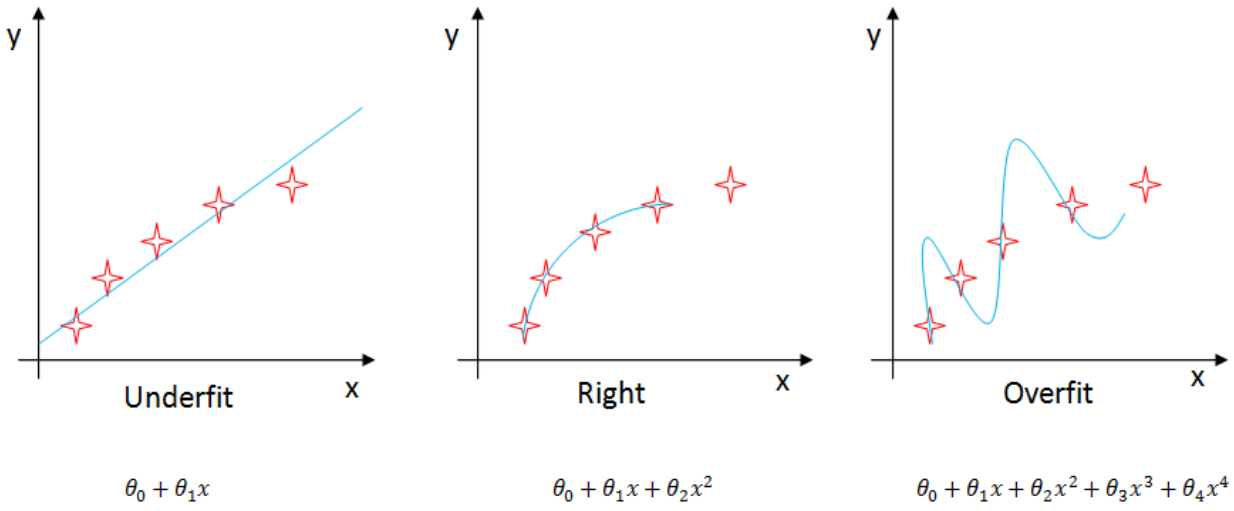

Figure 1.6 Underfitting, right fitting and overfitting training

The red crosses represent the data distribution. They follow a parabolic distribution. The blue line is the function that approximates the distribution. It can be considered the function learned by the artificial neural network. The expression under the plot is the mathematical expression of the function. In the first case the function is underfitting the data distribution. It means the model is too simplistic to accurately represent the data. In fact, the model is a line. The result of the model is that it inaccurately represent the data points and thus is not able to predict future data results as shown in Figure 1.7.

The underfitting problem can be avoided using a more complex model(e.g., a deeper neural network), improving the optimization algorithm or reducing the regularization (33) (if used).

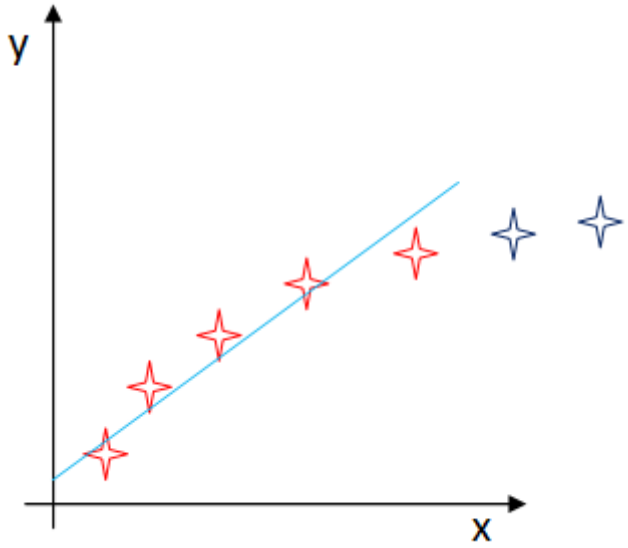

Figure 1.7 Underfitting phenomenon

The correct model is the one represented in the second case of the Figure 1.6. It is a parabolic model like the one used for the data distribution.

In the last case of the Figure1.6, the function is overfitting the data distribution. The overfitting is the result of an analysis that corresponds too closely to the training data and may therefore fail to fit to additional data like the validation or test data. Because of this the model may fail to predict future observations reliably. Generally, when a learning algorithm is more accurate in fitting known data (hindsight) but less accurate in predicting new data (foresight), it is said to be an overfitting algorithm (see Figure 1.8). Usually it happens when a more complicated approach than is ultimately optimal, is used. In the last case represented in the figure, a 4th degree equation is used to represent a 2nd degree distribution of data.

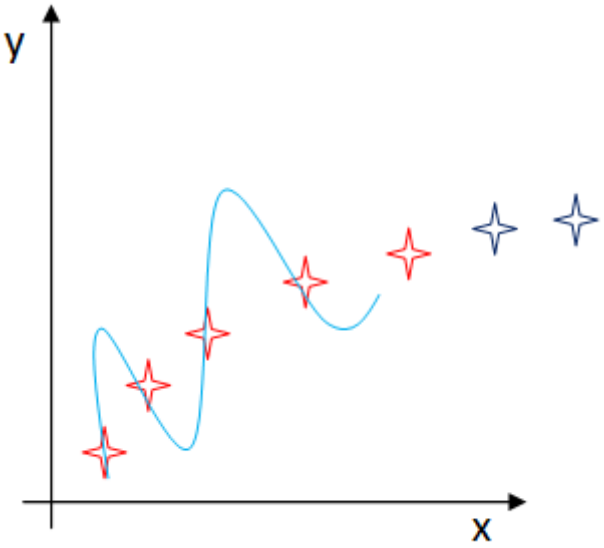

Figure 1.8 Overfitting phenomenon

Overfitting is especially likely in cases where learning was performed too long or where training examples are rare. This causes the learner to adjust to very specific random features of the training data that have no causal relation to the target function. In this process of overfitting, like already said, the performance of predicting training examples still increases while the performance on unseen data becomes worse, as schematically represented in Figure 1.8.

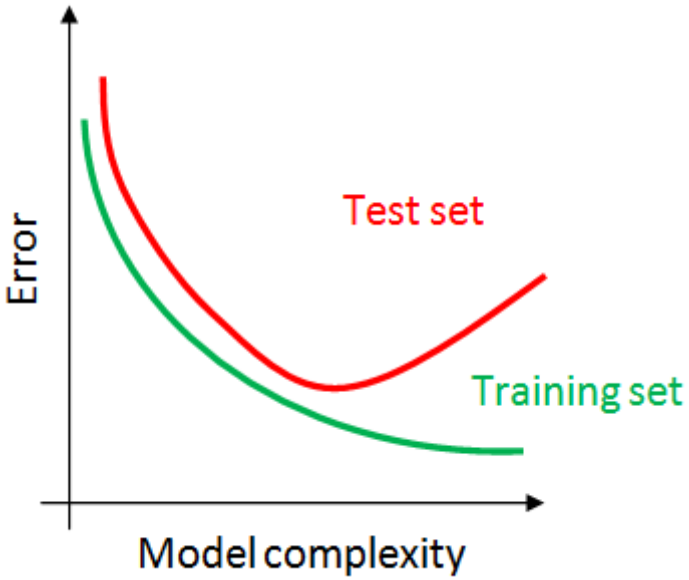

Figure 1.9 Overfitting phenomenon birth with the increase of model complexity

The overfitting problem can be avoided using more data for the training. Usually the size of the data set for the training process is about 10^{4-5}. In the case the data available are few and cannot be expanded, the overfitting can be avoided using a simpler model (Figure 1.9) or using fewer features. In the case none of these solutions can be adopted, or they are not effective, it is possible to add the regularization to the model (Figure 1.10). The regularization modifies the cost function adding a term that penalizes models with large values of parameters. It is added to the output signal considering the absolute value (L1) or the squared value (L2). In this study a L2 regularization is used. The L1 regularization is analog to the L2 one. In the following expression an L2 regularization is added for multivariate linear regression.

$$L(\Theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}(\Theta, \boldsymbol{x}_i) - y_i)^2 + \frac{\lambda}{2N}\sum_{j=1}^{k}\theta_j^2.$$

Note that the constant θ_0 (bias term) is not regularized. The regularization parameter is usually varied in log scale: $10^{-6}, 10^{-5}, \ldots, 10^{-2}, 10^{-1}$. If the regularization parameter is too small, it is like not having regularization. So, if the overfitting phenomenon is present, it will be maintained. If the regularization parameter is too high, the underfitting phenomenon is present. The overfitting phenomenon implies high variance. On the contrary, the underfitting phenomenon implies high bias.

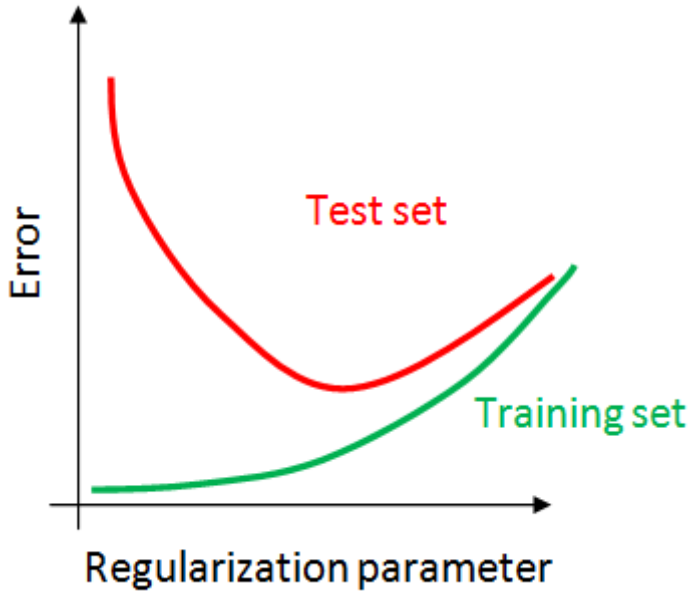

Figure 1.10 Overfitting phenomenon neutralization with the use of regularization parameter

1.3.4 Stopping criterion

The training process can be stopped mainly according to two criteria (34). The first criterion is based on a loss function. The monitoring of the loss function permits to check the trend of this function and to stop the training when it does not decrease any more. As anticipated, the loss function used in this study is the mean squared error. Usually a tolerance on the stopping criterion is applied, in order to avoid the training stopping for a momentaneous increase of the loss function. With the aim of avoiding overfitting, the stopping criterion is usually applied on a set of data not used for the training process. This is called the validation set.
The other stop criterion is simply the number of iterations. When the maximum number of iterations set is reached, the training stops

1.4 Neural network training strengthening: transfer learning

The transfer learning technique is a method thought to increase the network capability of learning training data, but also to minimize the resources used to train an artificial neural network (35) (36).
The term transfer learning is used to indicate an advanced machine learning method in which a model pre-developed to perform a generic activity, is used as a starting point for developing another one, aimed to perform a different activity. This method is often used in image classification.
The first article in which the transfer learning is explicitly treated was published in 1976 by the scientist Stevo Bozinovski e Ante Fulgosi. The research on transfer learning is continuing and now Andrew Ng, associate professor at Stanford University, co-founder and head of Google Brain, believes that the transfer learning will be the next driver for commercial success of machine learning.
As anticipated, the transfer learning consists in the training of an artificial neural network in which a pre-train on a big database has been already performed. The pre-trained model can be used as it is or can be used as a base for a personalization on other data and, so, to perform another activity. The idea behind the transfer learning is that if a model is trained on a sufficiently large and general database, the general maps of functionality learned can be used in analog tasks. This method permits to achieve

functionalities without the necessity of training from scratch a new artificial neural network on a database sufficiently large to guarantee a good result.
There are 2 main models to use transfer learning.
The first model is used to extract further functionality from the behaviour of an artificial neural network already trained and to apply it on new data. With this aim it is sufficient to add to the already trained model, a new classifier, which is trained from scratch. This method allows the reuse of the conceptual maps already developed in the first model without the necessity of training from zero a whole artificial neural network. This model of using transfer learning is applicable for solving similar problems.
The second model is the fine-tuning. It is used to limit the training to a sensible inferior number of data. In this model the more external layers of neurons of the pre-trained artificial neural network can be unlocked. These layers are further trained in a specific way for the final task. The fine-tuning allows the refinement of the basic model, obtained with the pre-training, in order to have a more performing model for the specific final task. In this way the training is faster and the resources requested are less ingent with respect to training of an artificial neural network from scratch.

Summarizing, the transfer learning allows the reuse of the largest part of the weights of an artificial neural network already trained for solving a similar problem. It limits the training to the layers dedicated to the classification or regression of the features already obtained in the previous layers. In this study we use this method to strengthen the model built, especially in the case of fine-tuning.

1.5 Neural network performance descriptors

The performance of an artificial neural network can be measured using many parameters, depending on the type of problem. In particular, for the regression problem, the performance are mainly measured using the parameters hereafter. In the following expressions, y^i is the i^{th} real data and $\hat{y}(\Theta, \boldsymbol{x}^i)$ is the i^{th} predicted data. N is the total number of data. $\boldsymbol{x}$ is the features matrix. Θ is the weight matrix.

- Mean squared error:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y^i - \hat{y}(\Theta, \boldsymbol{x}^i))^2.$$

The mean squared error measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value.

- Mean absolute error:

$$MAE = \frac{\sum_{i=1}^{N} \left| y^i - \hat{y}\left(\Theta, \boldsymbol{x}^i\right) \right|}{N}.$$

The mean absolute error is a measure the average of the absolute value of the errors between the estimated values and the actual value.

- Coefficient of determination:

$$R^2 = 1 - \frac{\sum_{i=1}^{N} (y^i - \hat{y}\left(\Theta, \boldsymbol{x}^i\right))^2}{\sum_{i=1}^{N} (y^i - \bar{y})^2}.$$

$\bar{y}$ is the mean value of the real values y^i.

The coefficient of determination is the proportion of the variation in the dependent variable that is predictable from the independent variables. If the model fit is perfect with real data, R^2 = 1. A baseline model, which always predicts $\bar{y}$ will have R^2 = 0. Models that have worse predictions than this baseline will have a negative R^2.

- Pearson's correlation coefficient:

$$R_p = \frac{\sum_{i=1}^{N} \left(y^i - \bar{y}\right)\left(\hat{y}\left(\Theta, \boldsymbol{x}^i\right) - \overline{\hat{y}(\Theta, \boldsymbol{x})}\right)}{\sqrt{\sum_{i=1}^{N} (y^i - \bar{y})^2} \sqrt{\sum_{i=1}^{N} \left(\hat{y}(\Theta, \boldsymbol{x}^i) - \overline{\hat{y}(\Theta, \boldsymbol{x})}\right)^2}}.$$

$\overline{\hat{y}(\Theta, \boldsymbol{x})}$ is the mean value of the predicted data.

Pearson's correlation (R_p) is defined as the ratio between the covariance of the data and the product between the standard deviation of the 2 variables. When applied to a sample, it is possible to obtain the shown formula by substituting the covariances and variances based on a sample. The correlation coefficient ranges from −1 to 1. An absolute value of exactly 1 implies that a linear equation describes perfectly the relationship between predicted and real data, with all data

points lying on a line. The correlation sign is determined by the regression slope. A positive value implies positive gradient for the line and viceversa for negative values. A value of 0 implies that there is no linear dependency between the variables. An absolute value of R_p inferior of 0.3 indicates a weak correlation between predicted and real data. An absolute value of R_p between 0.3 and 0.7 indicates a moderate correlation. An absolute value of over 0.7 indicates a strong correlation.

Many other performance descriptors for regression problem exist, but these are the most important for the present work.

For classification problems the measures of a test's accuracy mostly used are F-score, confusion matrix, receiver operating characteristic (ROC) and others.

2 Database

2.1 Database description

As discussed in the previous chapter 1.1.2, machine learning allows the machine to learn from experience. Experience, in machine learning, is synonymous with data. In fact, the machine makes its experience through the input data. The machine learning algorithms use computational methods to learn information directly from the data, without using a priori mathematical models or pre-determined equations.

The database is a fundamental element for machine learning. Without a sufficiently large database, the machine learning process cannot take place. For sure, the recent strong increase of machine learning applications in any possible field is due to the increased capability of collecting and storing an enormous quantity of data. The enormous quantity of data is contemporarily a cause, but also a consequence of machine learning development. In fact, modern hardware permits to collect large quantity of data in a small device. This fact encourages data storing also in common applications. The increasing collection of data determines the birth of the big data phenomenon. Big data refers to data sets that are too large or complex to be dealt with by traditional data-processing application software. As already seen, some applications of artificial intelligence are used in the analysis of big data (37) (38).

On the other hand, the promising results obtained in recent years by artificial intelligence applications, like machine learning, push the scientific community to create new large database containing any kind of data which can be used as starting point for machine learning applications.

This is the case of this study. In fact, in the next chapter we will discuss the difficulty in trusting the results coming from classic scoring functions. For this reason, a possible solution is found in applying the machine learning techniques to the field of scoring functions. With this aim, the process of measuring more and more binding structures and creating shared database containing data on chemical complexes has significantly increased in recent years. Anyway, we will further explore this aspect in the next paragraphs. For this moment, we consider how the database is used in machine learning. In particular we consider supervised machine learning applications because in this study we are dealing with these types of applications.

Typically the database is subdivided into two or three subsets in order to complete the training and testing processes for the artificial neural network. The subset are used for training the model, for testing it and possibly for validating the model during the training for early stopping or for hyper parameter selection. In supervised machine learning, the input data consist in data that describe instances and in a label/target-value:

$$\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}.$$

The data which describe instances belonging to the training set are used as input data for the artificial neural network. The label/target-values are the desired output for the related input and are used in the machine learning process. The training set is normally the largest subset. Typically a percentage of about 80% of the entire database is used. In our study a percentage of the entire database which varies between the 80% and the 93% is used for the training set. The variation depends on the type of database considered.

The test set is normally composed of about 20% of the data base. In our study it uses a percentage which varies between 20% and 7% of the entire database. The data which describe instances belonging to the test set are used as input in the test procedure of the model. The label/target-values are used as term of comparison with respect to the predicted values to measure the performance of the model. Data present in the training set are not present in the test set. Consequently the two subset do not overlap.

The validation set is a possible partition of the database which can be used in the monitoring process to determine the ending of the training phase. The dimension of the validation set is similar to one used for the test set, so it can vary between the 10% and the 7% of the entire database. If a validation set is introduced, the part of the database used in the training is reduced and it consists in the remaining database excluding test set and validation set. The validation set contains a number of complexes used to monitor the training process. For this reason, also the validation set do not overlap with the other subsets. In particular, the validation set is used to calculate the loss function during training. Different functions can be used as loss function. In this study, the mean squared error (MSE) is used as loss function. The stopping criterion considers the variation of the loss function. Once the loss function is no longer decreasing for a fixed number of consecutively iterations, the training is stopped. It is useful to monitor the loss function on a validation set and not on the training set, because its measure must be an

index of readiness of the model and not of the learning process on the training data. The schematic subdivision of the database is represented in Figure 2.1.

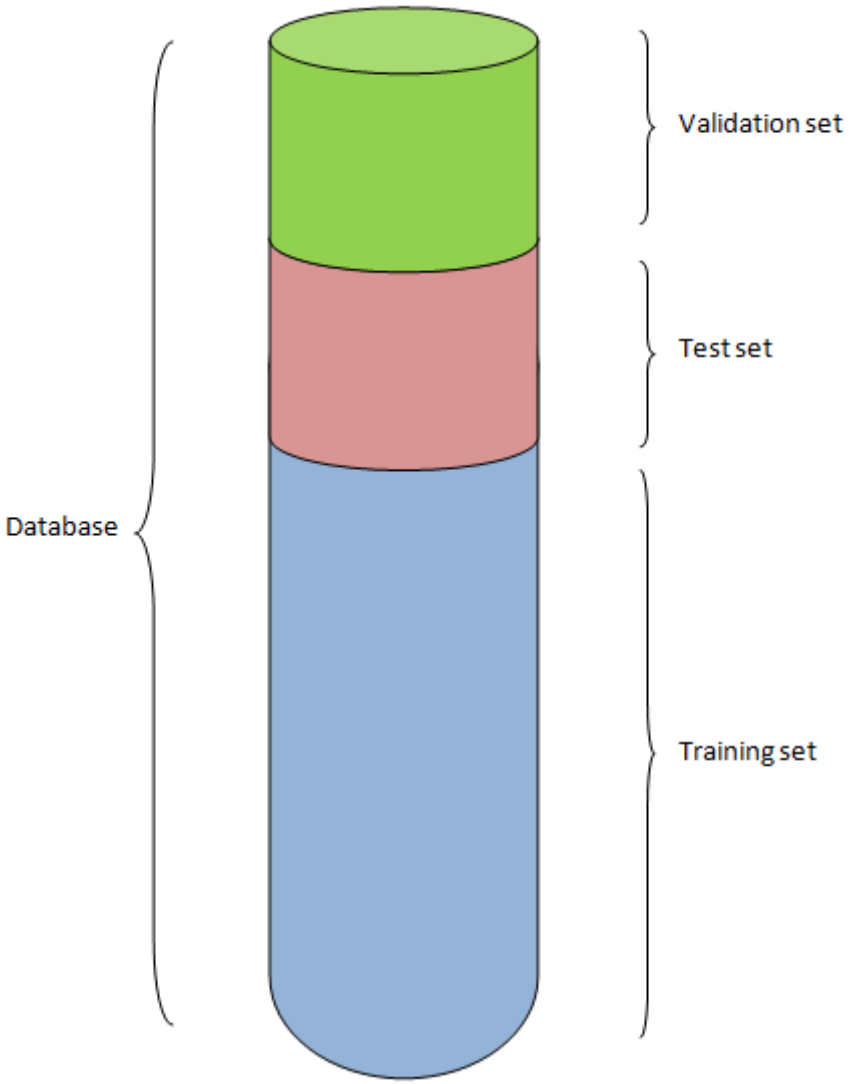

Figure 2.1 Schematical representation of the database and its subdivision in subsets

The data used in this research are ligand-protein atomic structures, associated to the corresponding affinity value or docking score.

2.1.1 Proteins

Proteins are polymers of 20 different amino acids (39). Proteins, in addition to some inorganic elements (for example, calcium and phosphorus, necessary for calcification bone), and some essential fatty acids (for example, linoleic acid, a part of cell membranes) carry out the plastic function of nutrients or rather that which allows the construction of new living matter. Proteins are the building blocks of living organisms.

This peculiar function, called plastic, however, is not the only one. In fact, proteins are also involved in the synthesis of hormones, enzymes, and tissues (especially muscle). In conditions of low energy intake, proteins, derived from food or muscle catabolism, can be used by the liver to provide energy to the organism.

As anticipated, from a chemical point of view, proteins are macromolecules made up of 20 fundamental units called amino acids, which, like many rings, join together to form a long chain (40). Proteins are composed of one or more polypeptides chains, i.e. linear compounds formed by amino acids linked one after the other (41). There are hundreds of amino acids, but only twenty of them are part of proteins. Anyway, they are a sufficient number to form more than 50,000 different proteins which are the ones present in the human body. Proteins are composed of as little as 50 amino acids to a maximum of a few thousands. The shorter chains are called oligopeptides (number of amino acids less than or equal to 10) and peptides (from 10 to 50 amino acids).

Eight of these amino acids are essential because the body does not manage to synthesize them fast enough to meet the metabolic demands. These amino acids are leucine, isoleucine, lysine, methionine, valine, threonine, phenylalanine, tryptophan. They must, therefore, be introduced with food, in order to avoid specific nutritional deficiencies. In the first two years of life two other amino acids become essential. They are called arginine and histidine respectively. For all the other amino acids there are enzymatic systems that make their endogenous biosynthesis possible.

Many foods are rich in proteins: meat, fish, but also the plant world offers a good sustenance: rice, wheat, maize, and sorghum are rich in these elements.

Each protein performs one or more functions in the body, generally through specific interaction with other molecules. The conformation (three-dimensional structure) of each protein determines the type of possible interactions and therefore the specific function. It depends on the sequence of amino acid residues contained in the protein. The abolition (denaturation) of the protein conformation determines the loss of function.

Among the most important functions exercised by proteins in organisms, there are: structural function (collagen); transport function (hemoglobin, apolipoproteins, albumin); defence and protection function (immunoglobulins, fibrinogen); control and regulation function (hormones, receptors of different hormones, transcription factors); catalytic function (all enzymes); movement function (actin, myosin).

The proteins can be classified according to their chemical composition in simple (composed of only amino acids) and complex (composed of amino acids and other

substances different from them like lipoproteins, glycoproteins, nucleoproteins, phosphoproteins.

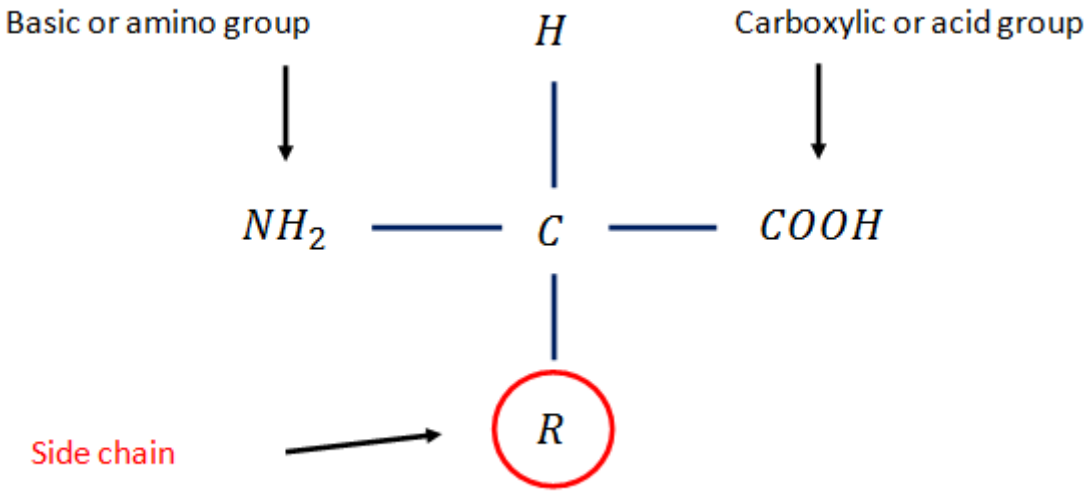

Figure 2.2 Amino acid basic structure

What essentially determines the role of an amino acid in a protein is the nature of the side chain indicated with the following symbol: -R (Figure 2.2). These functional groups are responsible for the structure, functions and electrical charge of proteins. Amino acids can be classified according to the properties of their side chains (-R), considering their polarity or non-polarity at the physiological pH and therefore the tendency to interact with water. Amino acids with charged, hydrophilic side chains are generally exposed on the surface of proteins. Non-polar hydrophobic residues are generally found within proteins, protected from contact with water.

The structural feature common to all proteins is that they are polymers of amino acids. For this reason amino acids are the structural elements of proteins. The protein molecule is a polymer consisting of amino acid monomers linked by the peptide bond (42). Amino acids are joined by the peptide bond in which the carboxyl group of one amino acid reacts with the amino group of another amino acid with elimination of a water molecule. More amino acids can join to form linear chains, in which both functional groups, basic and acid, of each amino acid residue are involved in peptide bonds. The formation of peptides results from the concatenation of multiple amino acids through amide (or peptide) bonds (see Figure 2.3). The peptide bond is rigid and planar. The binding energy is very high and the bond can be broken by boiling and prolonged action of strong acids or bases. Proteolytic enzymes can selectively break these bonds.

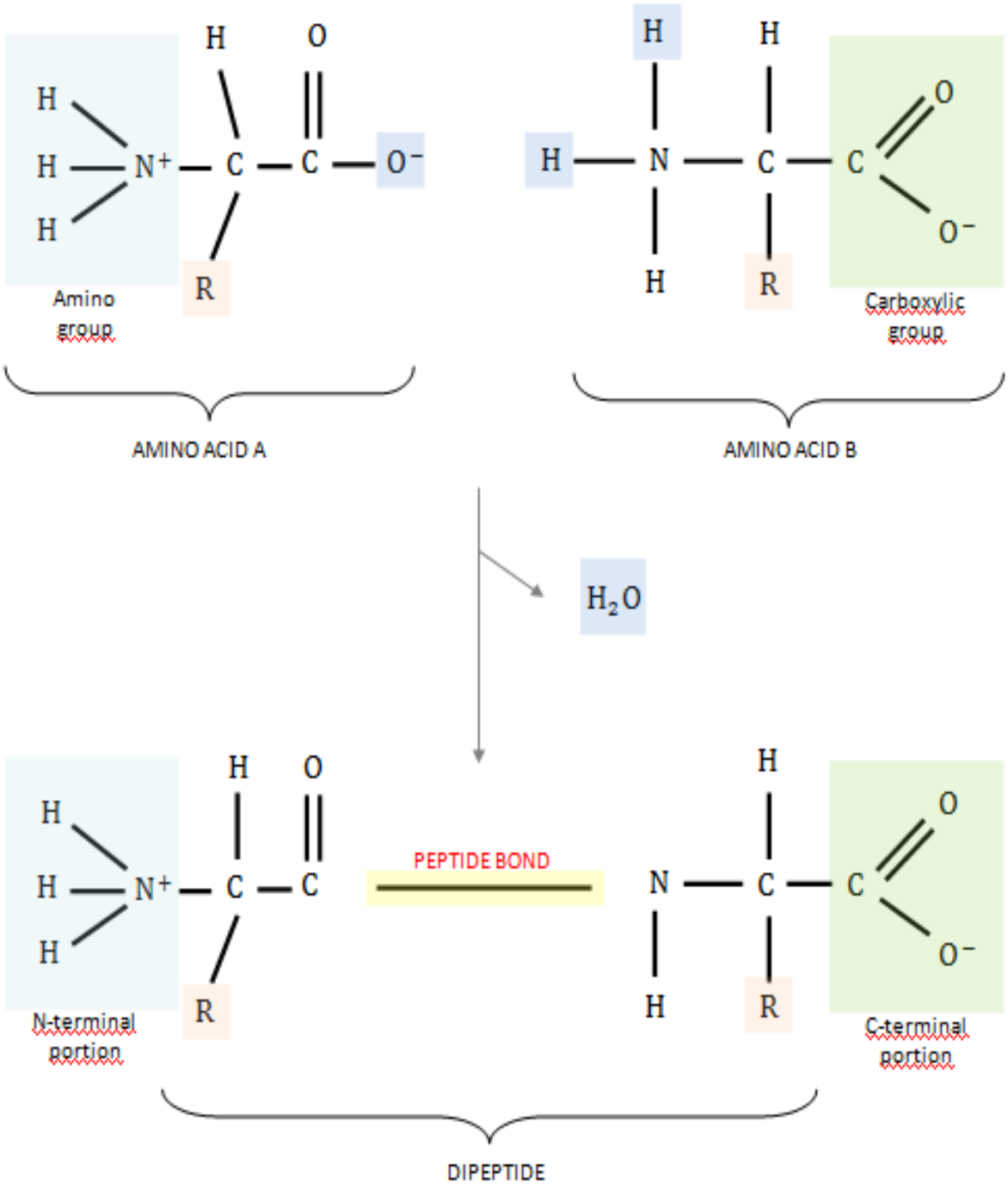

Figure 2.3 Formation of peptides results from the concatenation of multiple amino acids through amide (or peptide) bonds

The functional variety of proteins is determined by several factors: the number of amino acids; the type of amino acids; the way in which they are linked; the spatial arrangement; the shape of the polypeptide chains.

The levels of organization of proteins are the following (43):

• Primary structure: describes the amino acid sequence and is always linear.

• Secondary structure: describes the shape of the chain and can be helical or pleated.

• Tertiary structure: describes the three-dimensionality of the chain with attention to the local or remote relationships of the R groups and to the globular and fibrous proteins (collagen).

• Quaternary structure: describes the interactions between several protein chains.

The primary structure is given by the amino acid sequence in the polypeptide chain.

In the secondary helical structure, an H bridge is created for every 3.6 amino acids. The H bond is established between the H of the amide nitrogen and the O of the carbonyl group. There are no hydrogen bonds with the outside. The R groups project out orthogonally. The propeller is always right-handed.

In the folded secondary structure, different segments of the polypeptide chain, which have an extended arrangement, are parallel to each other (with an antiparallel course). The structure is stabilized by hydrogen bonds between the NH and CO groups of adjacent segments. The juxtaposition of different segments of the polypeptide chain gives rise to structures referred to as wavy beta sheets due to the bond angles.

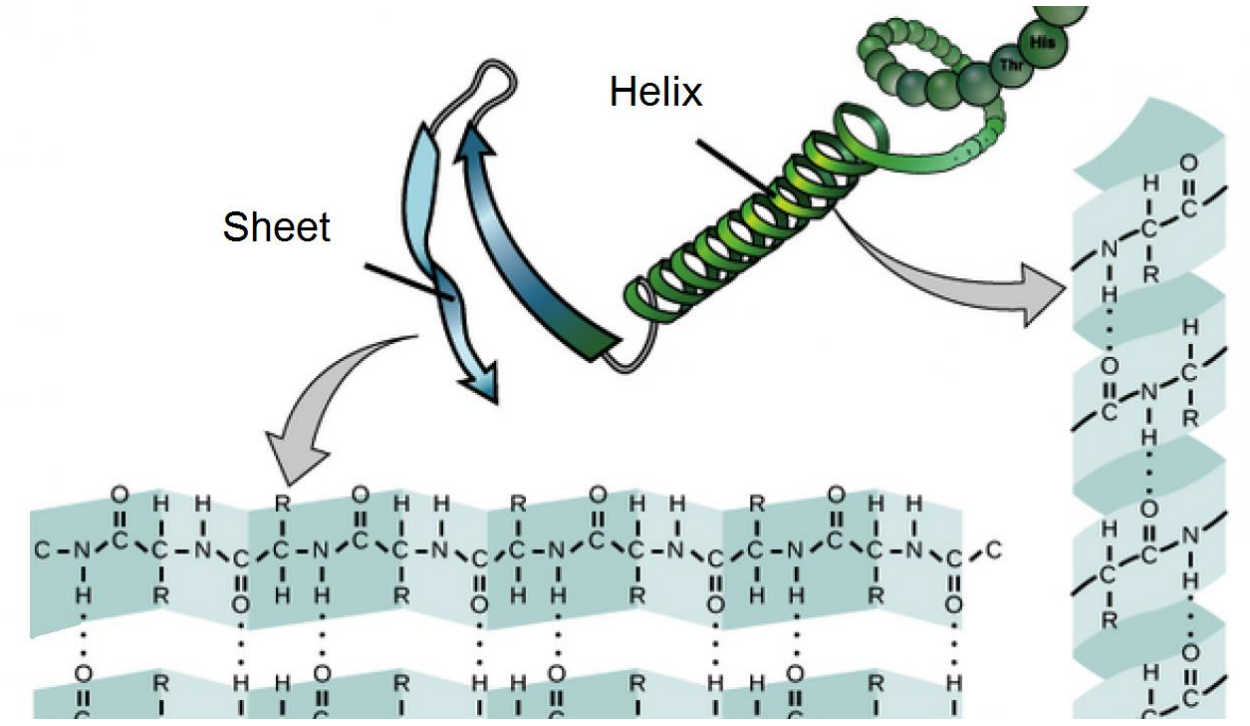

Figure 2.4 Helical and beta sheet regions

The tertiary structure is given by the combination of several helical and/or beta sheet regions connected to each other by segments that form loops. The looped regions

generally constitute the functional site of the protein. The tertiary structure is stabilized by secondary bonds that are established between the side chains of the amino acids. In some proteins we have a covalent bond, the disulfide bridge, which is established between two cysteine side chains. The tertiary structure is stabilized by hydrogen bonds between groups peptides, such as in the helix structure and in the beta sheets, hydrogen bonds between R groups, hydrophobic interactions between non-polar R groups, electrostatic bonds between R groups, R positively and negatively charged and disulfide bridges.

The quaternary structure foresees that the protein, formed by several polypeptide chains (subunits), is united with the same type of bonds that stabilize the tertiary structure.

2.1.2 Ligand

In chemistry a ligand represents an atom, ion or molecule that forms a coordination bond, generally donating its electrons. This bond is a type chemical bond in which two atoms jointly use one or more electrons, which was made available by a single atom, while the other makes use of the electrons shared by the first atom (44).

The molecule resulting from the union of one or more ligands is defined as a complex. The main factors that characterize ligands are represented by their size, charge, and chemical nature. In a complex, ligands can both stabilize the host molecule and condition its chemical and structural properties.

In biochemistry, a ligand is defined as a molecule capable of binding with a biomolecule and forming a complex capable of performing or inducing a biological function. Furthermore, the interaction between ligand and receptor alters the conformation (the three-dimensional structure) of the receptor itself.

In practice, the ligand in biochemistry is usually a molecule capable of binding, through a weak interaction, to a target protein. This interaction can be an ionic bond, a hydrogen bond, or a Van der Waals interaction. The implementation of a covalent bond, therefore irreversible bond, between the ligand and the target biomolecule is very rare in biological systems. Substrates, inhibitors, activators and neurotransmitters can be considered ligands. A receptor is a protein, transmembrane or intracellular, which binds with a specific factor, defined as a ligand, causing a conformational change in the receptor which results in the onset of a cellular response or a biological effect.

For this reason the study of medicines is based on the interactions between ligands and proteins. In that case the ligands are the active principle of the drugs.

2.1.3 Ligand-protein complex

As anticipated, the functions of many proteins require reversible binding with other molecules, commonly called ligands (45). A ligand can be of a different nature, even another protein. The result of this binding is a ligand-protein complex. In this field, the term docking is used to indicate the association of the ligand to the target biomolecule. Docking is usually reversible. The strength of the bond that is established between the ligand and the target biomolecule is called affinity. The transient nature of ligand-protein interactions is essential for life because it allows the body to respond quickly and reversibly to environmental and metabolic changes. A ligand binds to a region of the protein (binding site) that is complementary to the ligand in terms of size, charge, and hydrophobic character. The interaction is specific and is able to discriminate between thousands of molecules present in the vicinity of the binding site. A protein can have different binding sites for as many ligands. The ligand-protein binding is often accompanied by a conformational modification of the protein which makes the binding site more complementary to the ligand with a consequent strengthening of the binding (induced adaptation).

The ligand-protein interactions can be regulated by binding with other specific ligands that can cause structural alterations in the protein that modify the affinity and therefore the bond strength with the first ligand.

The conformational changes can be subtle (molecular vibrations and small movements) or more evident (displacements of parts of the molecule structure of several nanometers). As previously said, conformational changes are most often essential for protein function.

2.1.4 Ligand-protein complex structure

Ligand-protein complexes are held together by shared electrons or, less frequently, by covalent bonds. Such bonds are directional. It means that the atoms adopt specific positions relative to one another, to maximize the bond strengths and to minimize their energy. In addition, the binding with other specific ligands can cause structural

alterations due to the variation in the bond energy . As a result, each complex has a definite spatial distribution of its atom which determines its structure. This topic is studied in structural chemistry. It deals with determining how atoms combine in definite ratios and how this is related to the bond directions and bond lengths. The properties of molecules are related to their structures. For example, the water molecule is bent structurally and therefore has a dipole moment.

Theoretically, the structure of a molecule is determined by solving the quantum mechanical equation for the motion of the electrons in the field of the nuclei (called the Schrödinger equation). In fact, in a molecular structure the bond lengths and bond angles are those for which the molecular energy is the least. The determination of structures by numerical solution of the Schrödinger equation is highly resource consuming. For this reason other methods are preferred. In recent years the use of supercomputers and the increasing power in standard computers cause an increase in the use of Schrödinger equations.

Structural information in a molecule can be obtained using many experimental techniques. Microwave vibration-rotation spectra or neutron diffraction can determine the nuclear positions in a molecule. X-ray diffraction experiments are used for studying the electron cloud surrounding the nucleus in a molecule. X-ray crystallography is a technique of crystallography in which the image, produced by the diffraction of X-rays through the space of the atomic lattice in a crystal, is recorded and then analyzed to reveal the nature of the lattice. Typically, this leads to determining the material and molecular structure of a substance. In an X-ray diffraction measurement a crystal is mounted on a protractor and gradually rotated as it is bombarded with X-rays which produce a diffraction pattern of regularly spaced points. In addition, further information can be obtained by nuclear magnetic resonance techniques (NMR) or electron spin resonance. Eventually visual images of individual molecules and atoms can be produced thanks to the advances in electron microscopy performed in recent years.

Nowadays many projects of sharing on line ligand- protein structures and relative proper features exist. As anticipated, the aim of these projects is to join forces in order to collect as many data as possible. They are helpful for various computational and statistical studies, among which there are many artificial intelligence applications, on molecular recognition, drug discovery, and many more. Some examples of shared databases are Protein Data Bank, PDBbind, and CSAR (46) (47) (48) (49) (50) (51) (52).

2.1.5 Ligand-protein complex affinity and docking score

The label/target-values used in this study are the ligand-protein affinity and the docking score (53). Proteins work by interacting with other molecules. Molecules that interact reversibly with proteins, without been altered by interaction, are called ligands. Ligand-protein interactions are the basis of a very large number of biochemical processes. The quantitative description of these interactions represents one of the most relevant research topics of biochemistry. The key parameter used to describe these interactions is the dissociation constant, K_d. The K_d expresses the relationship that exists between the concentration of the ligand and the fraction of the binding sites of the protein that are occupied by the ligand. A ligand that has a high affinity for a protein has a very low K_d value. If the ligand binds the protein tightly, the ligand concentration required to occupy half of the binding sites is low and the value of K_d is also low. A ligand with a low affinity for a protein has a high K_d value.

There are many parameters which are used to measure the ligand-protein affinity. In our study we chose two similar parameters, one for experimental and one for synthetic data. They will be described in the next paragraphs.
The docking score is related to the molecular docking. Molecular docking, in the field of molecular modelling, is a method that predicts the preferred orientation of a molecule towards a second one when they bind together to form a stable complex. Knowledge of the preferred orientation can be used to predict the strength of a ligand-protein association or bond between two molecules using for example a scoring function. In particular, docking is frequently used to predict the binding orientation of a pharmacologically active small molecule to its target protein, so that the affinity and activity of this molecule can be predicted. In this case, the protein has its binding task and the best binding orientation of the small molecule must be found. This is the case we are dealing with in this research. The purpose of molecular docking is to simulate the molecular recognition process by a computer. The aim is, therefore, to obtain an optimized conformation simultaneously for the protein, the ligand, and their relative orientation such that the free energy of the system is minimized. Two approaches are particularly popular in the molecular docking community. One approach uses a technique that describes the protein and the ligand as complementary surfaces. The second approach simulates the docking process in which the interaction energies of the ligand-protein complex are calculated. The surface complementarity methods describe a series of properties that make binding between protein and ligand possible. These

properties include molecular surface or complementary surface descriptors. In this case the molecular surface of the receptor is described in terms of the surface area accessible to the solvent and the molecular surface of the ligand is described in terms of its binding surface. The complementarity between the two surfaces results from the description of the shape correspondence which can help to find the complementary docking position between the receptor and ligand molecules. Further approaches are the description of the hydrophobic properties of the protein observing the cyclical presence in the atoms of the main chain and Fourier shape descriptor technique.
Approaches based on complementarity of form are typically fast and robust. However they cannot usually accurately model movements or dynamic changes in the ligand-protein conformation, although recent developments allow these methods to investigate ligand flexibility.

Simulating the docking process is a much more complicated process. In this approach, the protein and the ligand are separated by a physical distance. The ligand finds its position in the active site of the protein after a certain number of movements in its conformational space. Movements mean rigid body transformations such as translations and rotations, but also internal changes in the ligand structure such as internal rotation and torsion. Each move in the conformational space of the ligand induces a total energy chage of the system and therefore after each move the total energy of the system is recalculated.

The first requirement for docking is the structure of the protein. As anticipated, normally the structure has been determined by techniques like X-ray crystallography or, more rarely, NMR of proteins. The structure of the protein and a database of potential ligands serve as input to the program.

Successful docking depends on the search algorithm and on the scoring function. The research space in theory consists of every possible orientation and conformation of the protein coupled to the ligand. However, in practice, with current computational resources, it is very difficult to exhaustively explore all the research space comprising all the possible distortions of a molecule (molecules are dynamic, they can have a set of conformational states) and all the possible orientations of the ligand relative to the protein, in a given level of granularity. Most docking programs in use involve a flexible ligand and some attempts to attach to a flexible receptor in the protein. Each position of the pair is called a pose.

The scoring function takes a pose as input and returns a number indicating the possibility that the pose represents a favorable binding interaction. Most of the scoring

functions are based on the estimation of the energy of a pose. Less is the energy, more favorable is the pose.

Docking is commonly used in the field of drug design for hit identification, lead optimization, and bioremediation. Hit identification is simply the use of docking combined with a scoring function for quickly monitoring large databases of potential drugs in silico, to identify the molecules most likely to bind to the target of interest. Lead optimization means the use of docking to predict where and in what relative orientation a ligand binds to a protein for creating more reactive and selective analogs. In the bioremediation, the ligand-protein docked can be used to predict pollutants that can be degraded by enzymes.

2.2 Experimental data

The experimental data are used to create a very accurate database, and, consequently, a scoring function which can better learn binding proprieties.

The experimental database contains ligand-protein complexes whose structures were obtained through X-Ray crystallography and deposited into the Protein Data Bank (http://www.rcsb.org).

The Protein Data Bank (PDB) is an open access digital data resource for biological experimental data central to scientific discovery. It was established as the 1st one in all of biology and medicine. The PDB archive was first announced in 1971 in *Nature New Biology.* The PDB provides access to 3D structure data for large biological molecules (proteins, DNA, and RNA). PDB stores an enormous wealth of 3D structure data. It has underpinned significant advances in understanding of protein architecture, culminating in recent breakthroughs in protein structure prediction. In fact, the PDB was created with the aim of helping in accumulating knowledge of 3D structure, function, and evolution of biological macromolecules, expanding the frontiers of fundamental biology, biomedicine, and biotechnology. The team who works on the PDB is called RCSB PDB (Research Collaborator for Structural Bioinformatics PDB) and it is served also by recognized experts in fields as advisors, including but not limited to, structural biology, cell and molecular biology, computational biology, information technology and education.

We selected a set of structures whose ligand-target interaction information is available from the PDBbind database (http://www.pdbbind.org.cn/). The PDBbind database provides a comprehensive collection of experimentally measured binding affinity data

for biomolecular complexes deposited in the Protein Data Bank (PDB). It provides an essential linkage between the energetic and structural information of those complexes, which is helpful for various computational and statistical studies on molecular recognition, drug discovery, and many more. The PDBbind database was originally developed by Prof. Shaomeng Wang's group at the University of Michigan in USA, and was first released to the public in May, 2004. This database is now maintained and further developed by Prof. Renxiao Wang's group at College of Pharmacy, Fudan University in China. The PDBbind database is updated on an annual basis to keep up with the growth of the Protein Data Bank.

The PDBbind has some subsets inside it. The PDBbind core set, one of these, aims at providing a relatively small set of high-quality ligand-protein complexes for validating docking/scoring methods. In particular, this data set has served as the primary test set in the popular Comparative Assessment of Scoring Functions (CASF) benchmark developed by the same group developing PDBbind.

2.2.1 Experimental data preparation

Among the complexes present in the database, we selected a set of structures with a resolution degree lower than 3Å and with a measured dissociation constant. The dissociation constant K_d is defined as:

$$K_d = \frac{[P][L]}{[C]}$$

where $[P][L]$ and $[C]$ represent the concentration of the protein, of the ligand, and of the complex, respectively.

In this study we use the pK_d value. The pK_d value is defined as:

$$pK_d = -\log_{10}(K_d)$$

It means we use only extremely detailed structures, in order to allow our model learning as many binding details as possible. This is an important condition we apply and not present in other studies on the subject. In fact, most of the studies do not apply particular conditions on the complexes used for building the model, apart from the nature of the data (experimental or synthetic).

Each structure was added of the hydrogen atoms with the help of molecular modeling and simulation software Molecular Operating Environment (MOE) (54) . MOE is a drug discovery software platform that integrates visualization, modelling, and simulations, as well as methodology development, in one package. Main application areas in MOE include structure-based design, pharmacophore discovery, fragment-based design, medicinal chemistry and biologics applications, molecular modeling and simulations, protein and antibody modeling, cheminformatics & QSAR. Eventually, a manual analysis is conducted on the complexes to check the rightness of the protein preparation.

For this reason, the experimental data are accurate, but not numerous. In fact, the process necessary to create an experimental datum is time and resources consuming. The total number of experimental data used in this study is 2400. In Table 2.1 the main characteristic of experimental data are resumed.

Type of data	Number of complexes	Meanligand-protein affinity
Experimental	2400	5.98

Table 2.1 Resuming table of experimental data

2.3 Synthetic data

For the synthetic database, we download a series of ligands for various targets which have the experimental 3D structure available, in the Protein Data Bank. The series of ligands were downloaded from the BindingDB online database (https://bindingdb.org). BindingDB is a public, web-accessible database of measured binding affinities, focusing chiefly on the interactions of proteins considered to be candidate drug-targets with ligands that are small, drug-like molecules. BindingDB supports medicinal chemistry and drug discovery in different ways: through literature awareness and development of structure-activity relations (SAR and QSAR). Through validation of computational chemistry and molecular modeling approaches such as docking, scoring, and free energy methods; through basic studies of the physical chemistry of molecular

recognition. BindingDB also includes a small collection of host-guest binding data of interest to chemists studying supramolecular systems. The data collection derives from a variety of measurement techniques, including enzyme inhibition and kinetics, isothermal titration calorimetry, NMR and radioligand and competition assays. BindingDB includes data extracted from the literature by the BindingDB project, selected PubChem confirmatory BioAssays and ChEMBL, for which a well defined protein target is provided. Data extracted by BindingDB typically includes more details regarding experimental conditions. BindingDB currently contains 2.096.653 binding data for 8.185 proteins and over 920.703 drug-like molecules (55).

2.3.1 Synthetic data preparation

In particular, we select compounds with an experimentally assessed data of affinity for the target, where the affinity is a K_i data. The K_i (inhibition constant) measures the affinity between a ligand and a protein using a reference radioligand. The parameter is very similar to K_d. We chose to use K_i for the synthetic data because the complexes available with this parameter measured are more numerous than the ones with K_d and because of the similarity between the two parameters. As for experimental data, we use $pK_i = -\log_{10}(K_i)$. Also for the synthetic data, we maintain a coherence in the affinity value used in the study. This is an important factor with respect to other studies on the subject where the type of the affinity value considered is not often specified. These series of ligands are then subjected to docking simulations at the respective target. The docking work is done with the help of GOLD docking engine through Molecular Operating Environment (MOE) software interface. GOLD stands for “Genetic Optimisation for Ligand Docking”. It is one of the most widely used docking programs and available as part of the CSD-Discovery and CSD-Enterprise Suites. GOLD’s evolutionary algorithm modifies the position, orientation, and conformation of a ligand to fit into one or more low energy states of the protein active site. It maps ligand geometry parameters onto populations of chromosomes and then runs evolutionary rounds of mutation, crossover, scoring, and selection to optimise ligand-protein interactions. The ligand structures are docked into the target binding site simulating a mean of 5 different poses and, for each ligand, the effective docking conformation is chosen as the one with the best score, according to MOE.

In the study, 17 different types of proteins are used as targets for the synthetic database. In the end, more than 100000 data are created by the docking software and the ones selected for the synthetic database are 28200. The proteins used are reported in the following table, where the quantity of respective complexes present in the database is also indicated. With this method it is much easier to create a big database because the process is completely automatic. However, the synthetic complex structure is not as reliable as the experimental one. In fact, the complex structure of synthetic data is, in general, created using docking software. As described in detail in the next chapter, in our study we use the MOE docking engine. A docking software uses a proper scoring function to determine the best configuration among different protein-ligand relative positions. The choice is based on the best score provided by the scoring function. However, as discussed in Chapter 3, the available scoring functions cannot produce accurate results, for various reasons carefully explained in that chapter. On the other hand, the experimental data include structures obtained via crystallographic radiography. The measured complex structure is assumed to correspond to the actual structure in their natural environment. This subject will be further discussed in Chapter 3. The database of synthetic data, for each ligand-target complex, contains not only the experimental affinity data K_i, but also the docking score produced by MOE software is registered. In Table 2.2 the main characteristic of synthetic data are resumed. In Table 2.3 the total number of complexes for each protein present in the synthetic database are reported.

Type of data	Number of complexes	Mean ligand-protein affinity	Mean docking score
Synthetic	28200	7.48	11.43

Table 2.2 Resuming table of experimental data

Protein	5HT2A	A2A	BACE1	DOP	FAAH	GR	H1	JAK1	PI3K
Complexes	2763	2914	1413	1243	508	843	1070	1213	1064
Protein	PIM2	ACE	KOP	M1	MCL1	JAK2	OX2	D2	
Complexes	384	488	2431	1056	688	1394	2160	6568	

Table 2.3 Complexes for each protein in the synthetic database

2.4 Data distribution

In the present paragraph, the characteristic of each database used in the research are discussed. In fact, each database used is different from the other. Synthetic data and experimental data consider different variables for target values, even if the ligand-protein affinity is considered in both cases, as previously explained. From the synthetic data, two database are extracted. They are using the same features, but they consider different target values: ligand-protein affinity and docking score. In this case the values have different physical meaning, but they are both supposed to measure the binding strength.

As discussed the experimental database is thought to be the principal database of this study. It is composed of 2400 data. The distribution of the target values considered is shown in Figure 2.5.

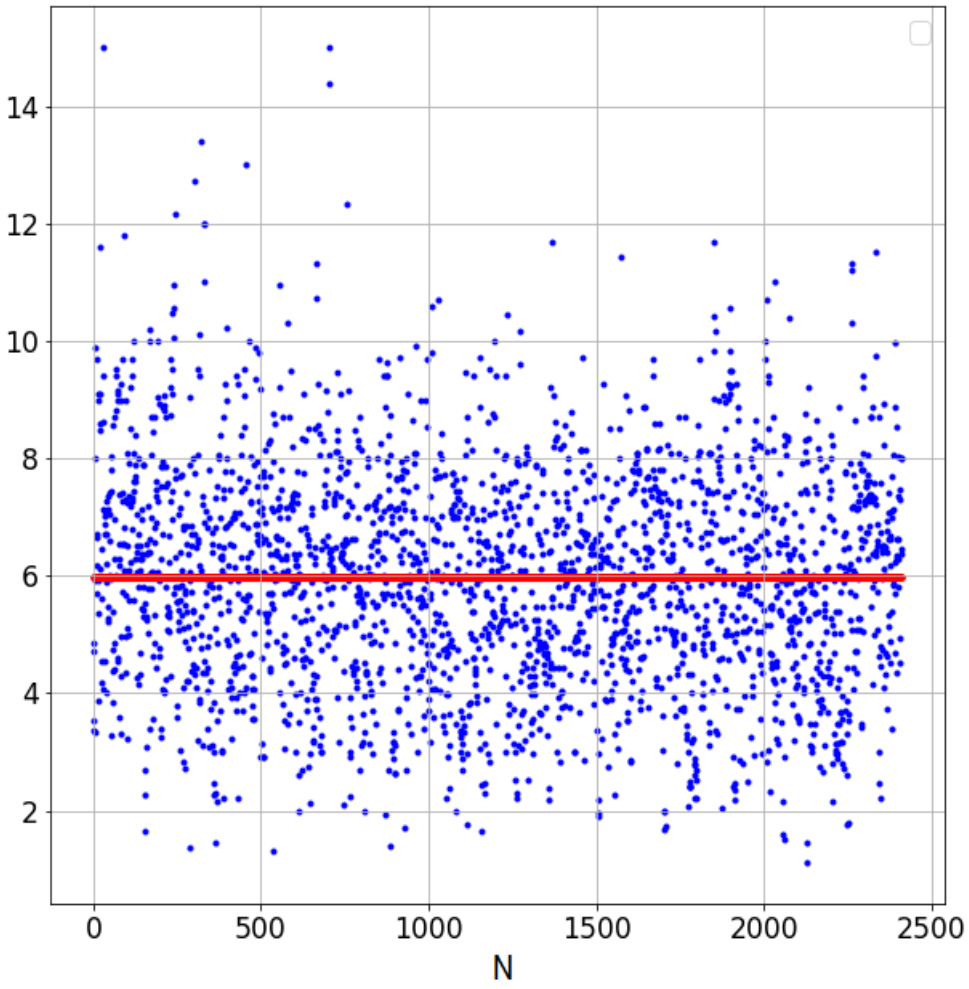

Figure 2.5. Experimental database, target values distribution

The mean of the ligand-protein affinity value in the whole dataset is 5.98. No outliers data are present in the set. An outlayer is a datum which is far from the mean value of the considered data distribution, This is a consequence of the data creation process, which involves checking the data one by one before inserting it in the database.

The synthetic database contains 28200 data, when the ligand-protein affinity is considered, the data distribution of the target values is the one shown in Figure 2.6. The mean of the ligand-protein affinity value in this whole dataset is 7.48. The difference between the mean value of ligand-protein affinity in experimental and synthetic data can be justified by the difference in the physical value considered between the two sets of data. In fact $K_d$and K_i are similar, but not exactly the same parameter, as already

discussed. The comparison between the target values of these two set of data is shown in Figure 2.10.

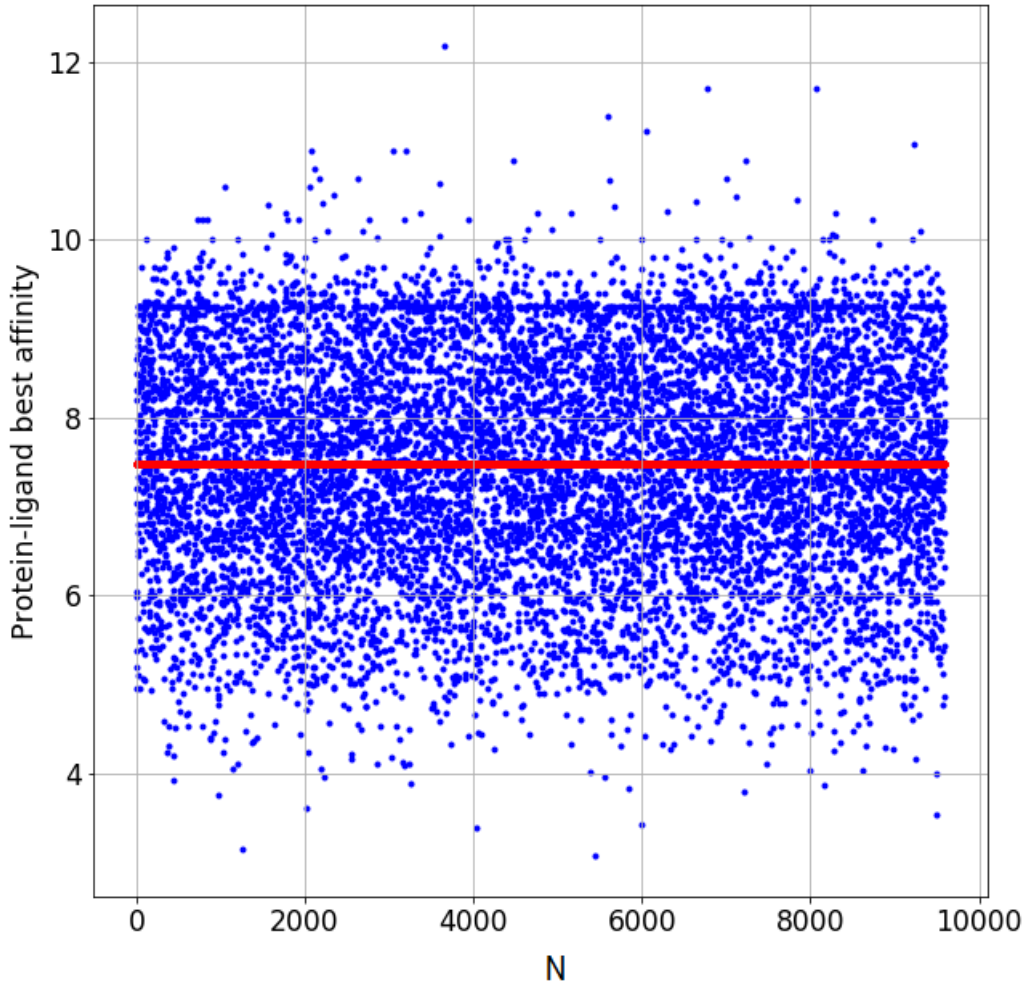

Figure 2.6. Synthetic database, target values distribution considering ligand-**protein affinity**

Eventually, also the docking score can be considered as target value for the synthetic data. The docking score considered is a negative value. The more negative the docking score is, the higher is the probability to have that pose for the complex. For the present study, the docking score is taken positively in order to have a distribution similar to the other target values. It means the original docking score is multiplied by -1. In this configuration, the higher the docking score is, the higher is the probability to have that pose for the complex. This is similar to the other target values, for which the higher the value is, the higher is the probability to find that configuration. The target value distribution of synthetic data with the docking score is shown in Figure 2.7. The mean of the docking scores converted for the data present in the database is 11.43. In the synthetic database, both when ligand-protein affinity or docking score are considered as

target value, some outliers are present. They are excluded from the data in the training and test phase of the scoring function.

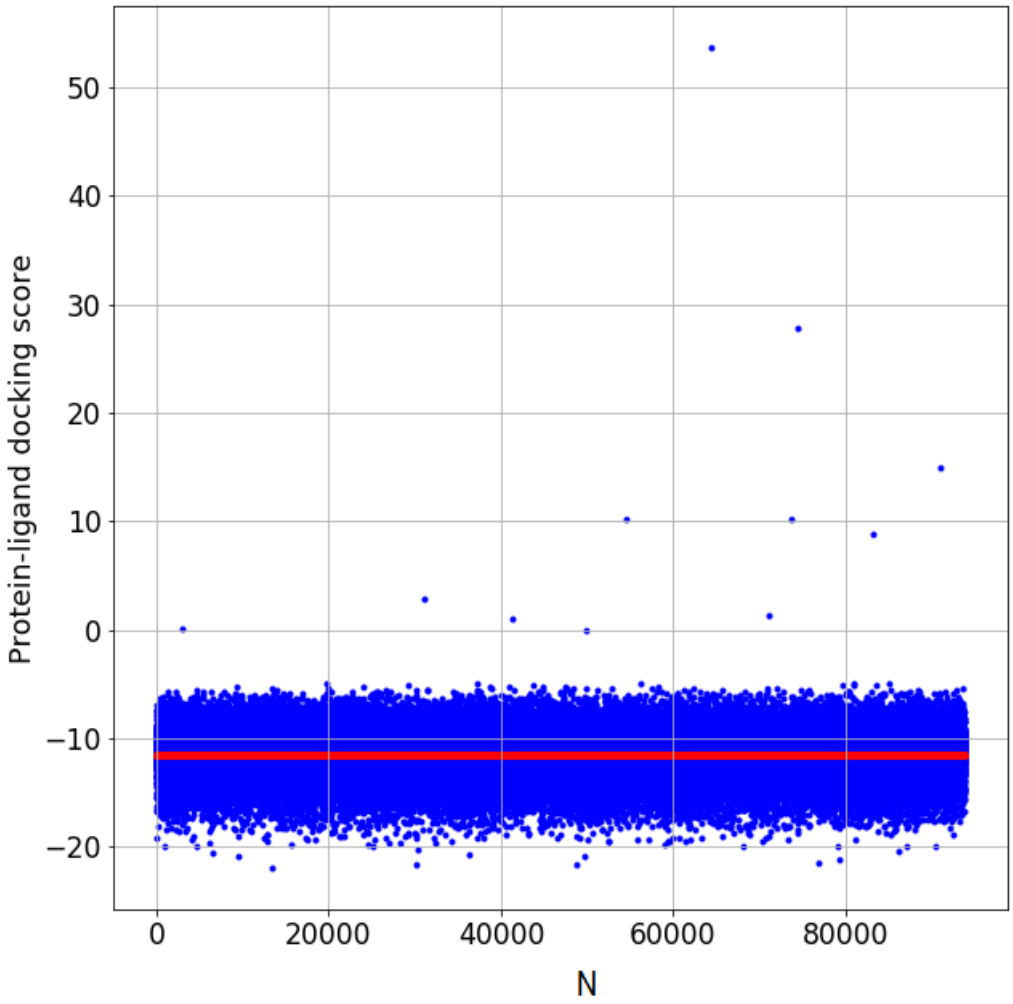

Figure 2.7. Synthetic database, target values distribution considering docking score

The difference of target values distribution when docking score or ligand-protein affinity, both if synthetic or experimental, are considered is more sensible. Actually, as anticipated in this paragraph, the total amount of synthetic data is more than 100000. Only the pose with the best score is considered in the synthetic database. Even if the docking score and ligand-protein affinity are different data, a comparison between the two, for each complex, is made to discover if some relation between the two target values is present (Figure 2.8). It is possible to see, in particular from Figure 2.9, that the two data have no evident relation. Only a similar trend can be observed. In particular, when the affinity decrease, also the mean value of the docking score has a decreasing trend.

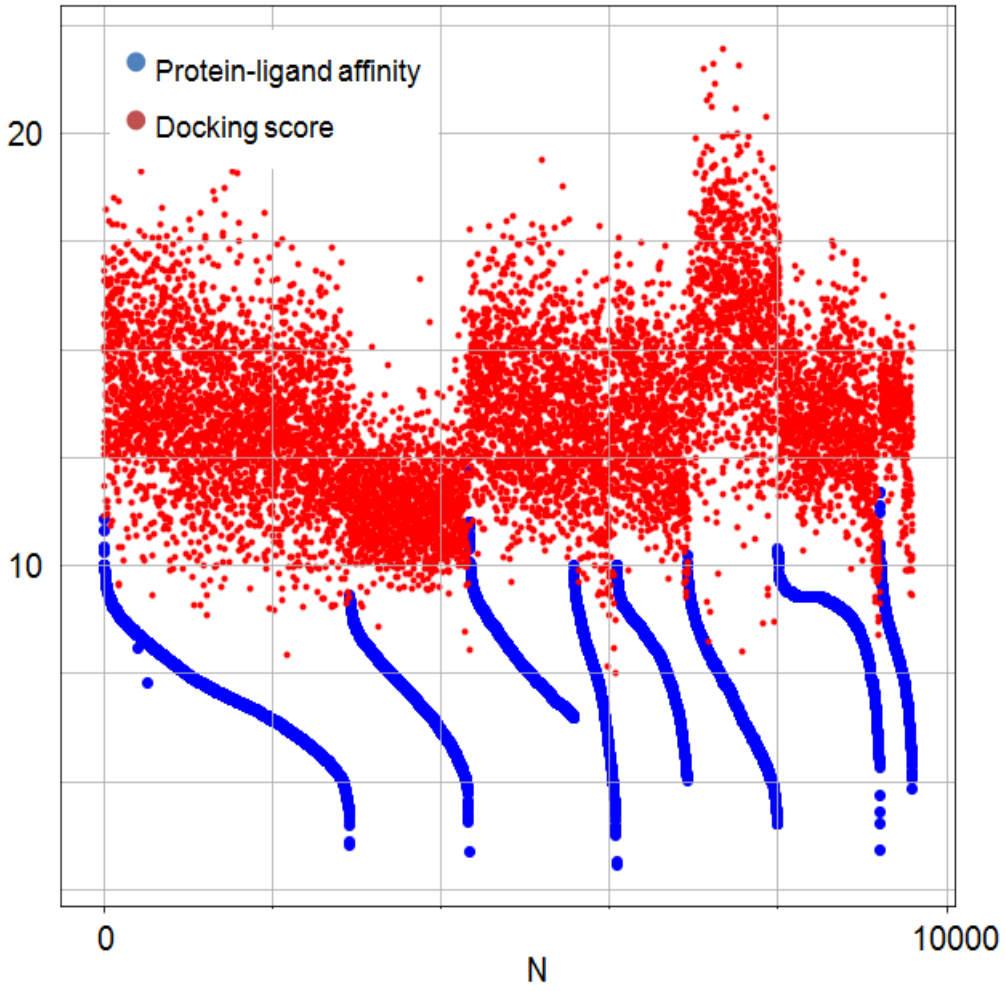

Figure 2.8. Synthetic database, target values distribution considering ligand-protein affinity and docking score contemporary. Different group of data are related to different targets

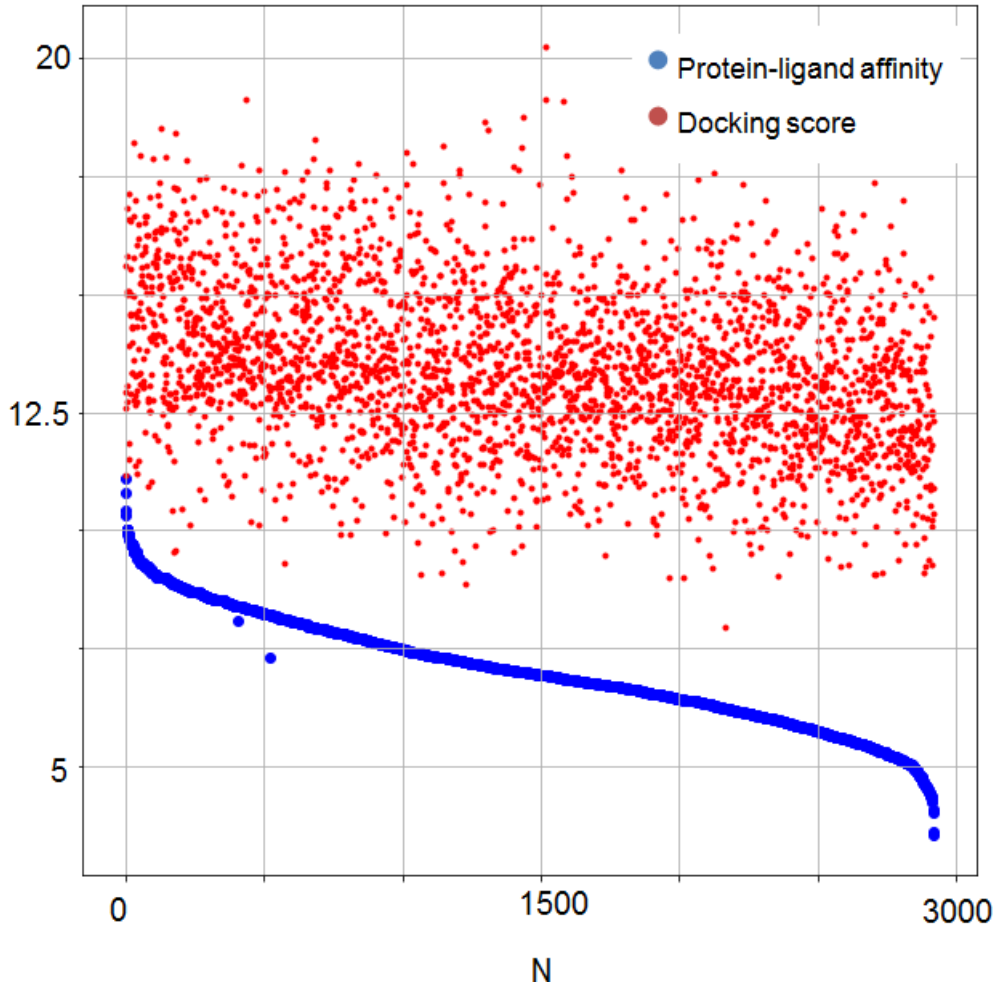

Figure 2.9. Synthetic database, target values distribution considering ligand-protein affinity and docking score contemporary for protein A2A

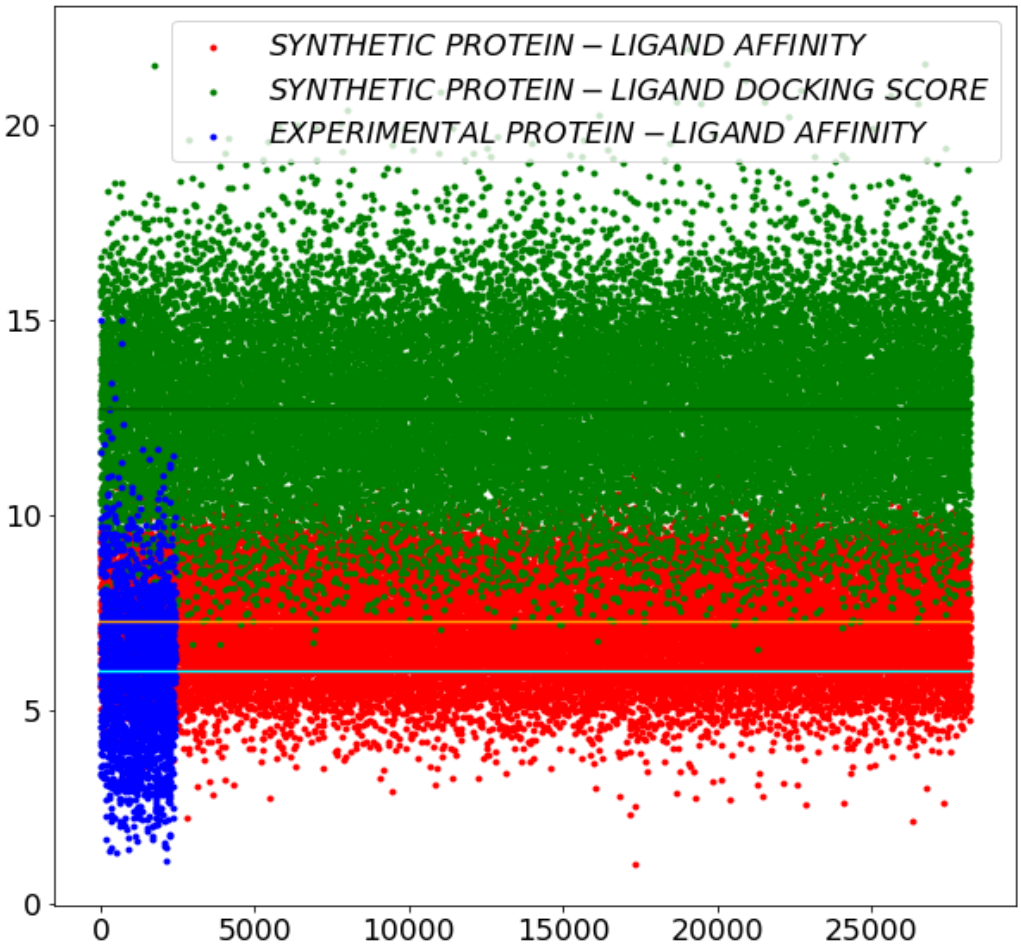

Figure 2.10. Database comparison according to the target values considered. The straight lines represent the mean value of each group of data

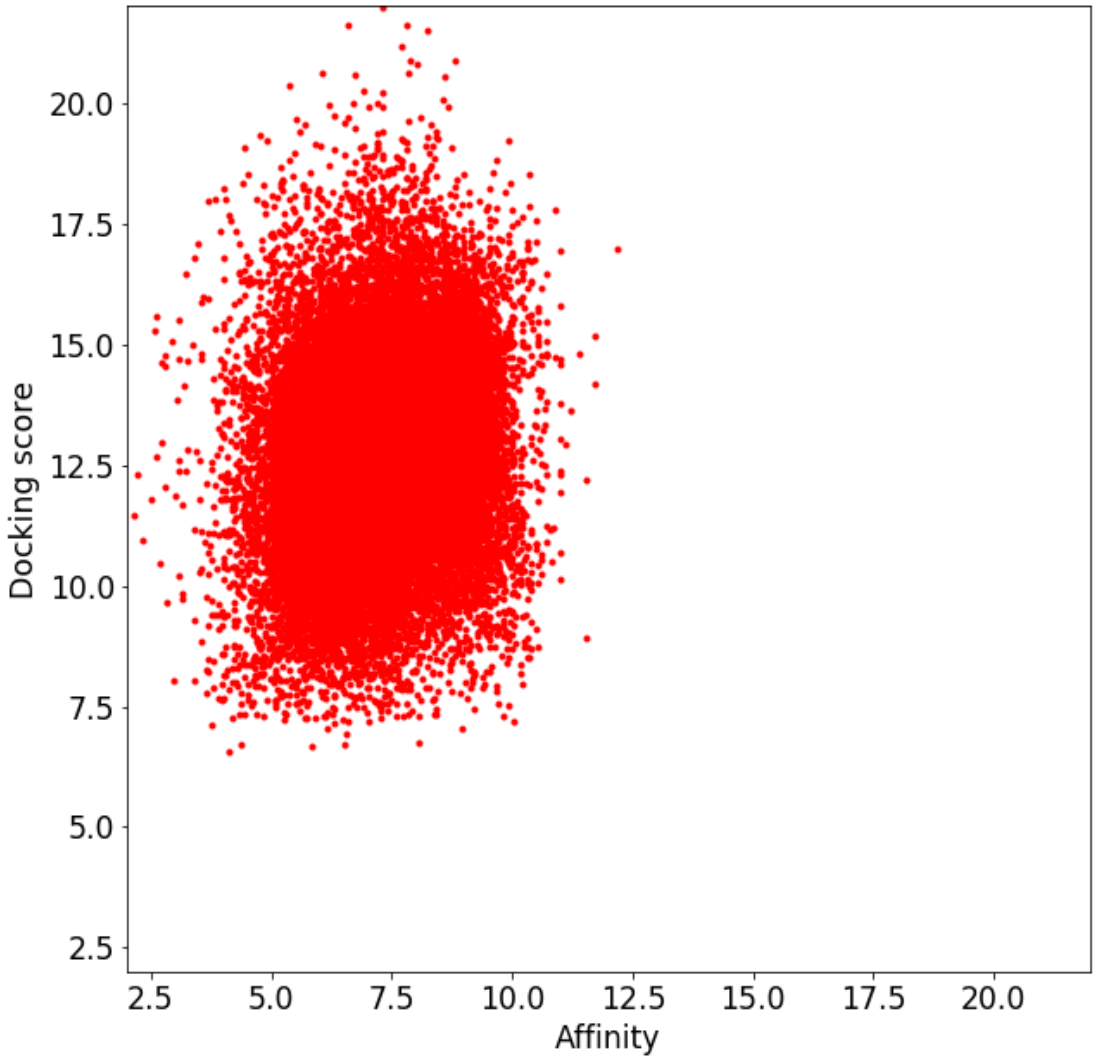

Figure 2.11. Comparison between ligand-protein affinity and docking score for the synthetic data. The Pearson correlation coefficient between these two target data is R_p = 0.17

2.5 Database creation result

The result of the database construction procedure both for experimental and synthetic data, is a list of .pdb file and a .xlsx file. The .pdb files contain the 3D structures of each complex. The .xlsx files contain the ligand-protein affinity and docking score (when present) of each complex. The .xlsx is a standard Excel file in which the name and the identification code of the complex are reported in two distinct columns and the relative targets values are reported in the adjacent columns.

```
ATOM      3  C    GLU A   2      34.642  24.919  30.006  1.00 35.15           C
ATOM      4  O    GLU A   2      33.523  25.105  29.533  1.00 35.13           O
ATOM      5  CB   GLU A   2      35.426  24.076  32.219  1.00 43.38           C
ATOM      6  CG   GLU A   2      34.388  24.808  33.049  1.00 52.75           C
ATOM      7  CD   GLU A   2      34.885  25.108  34.451  1.00 61.32           C
ATOM      8  OE1  GLU A   2      36.033  25.598  34.579  1.00 64.03           O
ATOM      9  OE2  GLU A   2      34.131  24.850  35.421  1.00 62.19           O1-
ATOM     10 1H    GLU A   2      35.961  21.922  30.664  1.00 42.26           H
ATOM     11 2H    GLU A   2      36.823  23.245  30.196  1.00 42.26           H
ATOM     12 3H    GLU A   2      35.657  22.642  29.221  1.00 42.26           H
ATOM     13  HA   GLU A   2      33.987  23.129  30.947  1.00 39.62           H
ATOM     14 1HB   GLU A   2      35.703  23.166  32.753  1.00 43.38           H
ATOM     15 2HB   GLU A   2      36.317  24.690  32.131  1.00 43.38           H
ATOM     16 1HG   GLU A   2      33.486  24.194  33.097  1.00 52.75           H
ATOM     17 2HG   GLU A   2      34.132  25.749  32.557  1.00 52.75           H
ATOM     18  N    ARG A   3      35.648  25.775  29.833  1.00 27.43           N
ATOM     19  CA   ARG A   3      35.477  26.995  29.049  1.00 21.30           C
ATOM     20  C    ARG A   3      35.184  26.662  27.584  1.00 20.36           C
ATOM     21  O    ARG A   3      34.417  27.363  26.929  1.00 18.75           O
ATOM     22  CB   ARG A   3      36.729  27.874  29.144  1.00 17.09           C
ATOM     23  CG   ARG A   3      36.888  28.553  30.485  1.00 17.66           C
ATOM     24  CD   ARG A   3      37.715  29.809  30.378  1.00 15.09           C
ATOM     25  NE   ARG A   3      37.851  30.481  31.667  1.00 14.40           N
ATOM     26  CZ   ARG A   3      38.696  30.105  32.625  1.00 17.55           C
ATOM     27  NH1  ARG A   3      39.484  29.057  32.444  1.00 15.93           N
ATOM     28  NH2  ARG A   3      38.755  30.778  33.766  1.00 18.59           N1+
ATOM     29  H    ARG A   3      36.534  25.630  30.313  1.00 27.43           H
ATOM     30  HA   ARG A   3      34.611  27.555  29.413  1.00 21.30           H
ATOM     31 1HB   ARG A   3      37.620  27.294  28.904  1.00 17.09           H
ATOM     32 2HB   ARG A   3      36.633  28.657  28.391  1.00 17.09           H
```

Figure 2.12. Content of a .pdb file

In Figure 2.12 a part of a typical .pdb file is shown. Each atom belonging to the complex is described by a line of the .pdb file. For this reason, the .pdb file contains a number of lines equal to the number of atoms composing the complex described. The following informations relative to each element are present in the file:

- Column 1: Membership molecule.
  The elements are divided in two groups, elements belonging to the protein (ATOM) and elements belonging to the ligands (HETATM).

- Column 2: Sequential number of atoms considered.

- Column 3: Atomic role.

- Column 4-5: Membership amino-acid.

- Column 6: Sequential number of amino-acid considered.

- Column 7-8-9: Cartesian coordinates.

The x, y, z coordinates are reported in column 7, 8, 9, respectively. The coordinates are in units of Ångströms [Å].

- Column 10-11: Occupancy and temperature factor.

- Column 12: Atomic species.

The .pdb file contains some additional information at the beginning and at the end of the file which are not useful for this research.

3 State of the art

3.1 Machine learning scoring function

In this chapter the argument of scoring functions is introduced and an excursus on the subject of machine learning scoring functions, starting from the origin until the present day, is made.

In medicinal chemistry and in pharmacology, scoring functions are used to accurately rank molecules based on their predicted affinity for a target of interest after they have been docked. Usually, one of the molecules is a small organic compound, such as a drug, and the second is the drug's biological target, such as a protein receptor. Anyway, the two molecules involved can also be two proteins or a protein and DNA. In this study, we will focus on the binding between a protein and a small organic compound.

The scoring function aims at describing the electrostatic, hydrophobic, solvation and hydrogen bonding interactions between the two molecules. The first goal is to discriminate between binders and non-binders. In fact, for example, the success rate of the initial phases of drug discovery depends on the prediction of the affinity of a candidate ligand for a therapeutic target (e.g., protein) of interest. The number of synthetically accessible small molecules is extremely vast. This combinatorial explosion underscores a core challenge in drug discovery: testing the affinity of as many small molecules as possible while maintaining a sufficient degree of accuracy. In fact, computationally exploring this entire space is currently intractable . There is, actually, a significant trade-off in both experimental and computational drug screening approaches between speed, cost, and accuracy (56) (57) (58). Chemists often use scoring functions also to rank poses of a ligand in a task (after they have been docked using a docking tool) and to predict the binding affinity of candidate ligands to a target protein (scoring power). This latter remains one of the most important and difficult incompletely solved problems in computational biomolecular science, and it is the one we are dealing with in this study.

Popular docking tools include a proper scoring function belonging to the classical type. Some examples are, e.g., GOLD (59), SurFlex Dock (60), or AutoDock Vina (56).

Classical scoring functions can be created using different types of approaches: empirical, force field, or knowledge based.

The empirical scoring functions are based on counting the number of various types of interactions between the two binding partners (61). They calculate the free energy of binding as a sum of contributing terms. Each term consists in a physicochemically distinct contribution to the binding free energy, such as: hydrogen bonding, van der Waals interactions, hydrophobic interactions, and the ligand's conformational entropy. Each of these terms is multiplied by a coefficient and the resulting parameters are used for estimating the binding affinities.

The force field scoring functions consider the strength of intermolecular van der Waals and electrostatic interactions between all atoms of the two molecules in the complex and, sometimes, also the strain and the desolvation energy. They parameterize the potential energy of a complex as a sum of energy terms arising both from bonded and both from non-bonded interactions (62). Each of these terms has a functional form characteristic of the particular force field. On the other hand, each force field contains a number of parameters that are estimated from experimental and simulated data. However, these force fields do not account for entropy because they were just designed to model intermolecular potential energies (63).

The knowledge based scoring functions are based on the following assumption. If a certain type of interaction between functional groups or atoms are encountered more often than expected by a random distribution, they should be energetically favorable, namely, they contribute favorably to binding affinity (64). They are called knowledge-based scoring functions because they use the 3D coordinates of a large set of ligand-protein complexes as a knowledge base. In this way, a ligand-protein complex model can be created on the basis of how similar its features are to those in the knowledge base. The features used are often the distributions of atom–atom distances between ligand and protein in the complex. Recurrent features in the knowledge base means favorable conditions, whereas less frequently observed features score unfavorably. The resulting score, deriving from the sum of these contributions over all pairs of atoms in the complex, is converted into a pseudo-energy function, typically through a reverse Boltzmann procedure, in order to provide an estimate of the binding affinity (65) (66) (67). Some knowledge based scoring functions nowadays include parameters that are fitted to experimental binding affinities (68) or introduce Information Theory-driven improvements as well as explicit solvent models (69).

In addition to scoring functions, there are other computational technique that provide a more accurate prediction of binding affinity, such as those based on molecular dynamics simulations. However, these techniques imply expensive calculations. For this

reason they remain impractical for the evaluation of large database of proteins or ligands and they are currently typically limited to family-specific simulations (62) (70).

Classic scoring functions can sometimes obtain good results in virtual screening experiments (57) (71) (72) (73). However, researchers working in medicinal chemistry need more consistent and reliable predictions, meaning that novel approaches are required. In fact, scoring functions do not fully account for a number of physical processes that are important for molecular recognition, which in turn limits their ability to select and rank small molecules by computed binding affinities. It is known that, among other drawbacks, classic scoring functions do not account well for solvation energy contributions or conformational entropy (74). Furthermore, it is generally believed (70) that two of the major sources of error in scoring functions are the implicit treatment of solvent and their limited description of protein flexibility. In addition to these enabling simplifications, there is an important issue that is often neglected, whatever type of scoring function is considered. The scoring function assumes a predetermined theory-inspired model for the relationship between the features that characterize the complex and its predicted binding affinity. However, the types of complexes present in nature can be extremely numerous and all of these do not conform to the rigid approach assumed in classic scoring functions. This inherent problem leads the scoring functions to poor predictivity in those complexes that do not conform to the modelling assumptions. For instance, the van der Waals potential energy of non-bonded interactions in a complex is often modelled by a Lennard-Jones 12-6 function with parameters calibrated with experimental data. In fact, the model for the relationship between the features that characterize the complex and its predicted binding affinity is often built based on experimental and simulated data. However, there could be many cases for which this particular functional form is not sufficiently accurate. Furthermore, while the r^{-6} attractive term can be shown to arise as a result of dispersion interactions between two isolated atoms, this does not include the significant higher order contributions to the dispersion energy, as well as the many-body effects that are present in ligand-protein interactions (75).

To rank or evaluate ligands from chemical libraries, the use of more than a single scoring function is a standard procedure. This guarantees a cross validation in the virtual screening performance. Often, before ranking a ligand-protein complex, an empirical or a knowledge-based or a force field function is used to generate an ensemble of possible docking poses. Each docking pose is further evaluated by an knowledge based scoring function. The results are used to correct the rank of the ligand

poses. However, resampling strategies, such as cross-validation or bootstrapping, are still not systematically used to guard against the overfitting of calibration data in parameter estimation for scoring functions (76).

In addition, a single classic scoring function can perform well or not depending on the target protein being addressed. This is a factor of uncertainty in classic scoring function results since the predictive accuracy of those functions varies between protein families. For this reason, scoring functions calibrated for the target under study are sometimes preferred to universal ones (77) (78). Importantly, the underlying often linear regression model employed by classical SFs has been shown to be unable to assimilate large amounts of structural and binding data (79). Still, it is worth mentioning that a number of control parameters can be adjusted to tailor the scoring function to a particular target and to select the major interaction type to be taken into account. However, most classic scoring functions cannot be trained on a particular target and are provided in a way that does not permit changing the regression model.
It is clear that the use of classic scoring functions can be laborious and not reliable, as just described, and it limits their use in large libraries of compounds.

In recent years, machine learning techniques are being applied in multiple research fields obtaining very promising results, in particular when the study has access to a very large number of data to learn from. This is often the case in the docking and in the virtual screening researching areas. Following this trend, the machine learning methodology has made its appearance in the docking and in the virtual screening pipelines.
It is necessary specifying that machine learning models appeared in virtual screening and chemoinformatics in recent years, not earlier than 20 years ago. Because of this, the present study and all the cited ones are pioneristic studies.
Here we focus mainly on algorithms posed as a regression problems for predicting ligand-protein binding affinities, even if they already appear in many other applications.

Machine-learning scoring functions provide clear advantages over classic ones (80) (81) (82) (83). They are sometimes several orders of magnitude faster than classic scoring functions. However, their performance can significantly vary depending on the model used and consequently on the chosen featurization (80).

These functions are created using different types of regression models, like random forests, logistic regression, support vector machines (SVM), or deep learning algorithms

trained on shared databases. A shared database is a database created by one or more research teams and shared on the web to have an amount of validated and reliable data available for any kind of research purpose. The binding affinity prediction with machine-learning scoring function interests many fields. Various outstanding open issues are being investigated by researchers. Which machine learning method could generate more predictive scoring functions (84)? How can one build machine-learning versions of classic scoring functions (85)? What is the impact of structure-based feature selection on predictive performance (86)? How does target diversity affect predictive performance (87)? How does predictive performance increase with the size of the training data, in both classic and machine learning types of scoring functions (79)? How does the quality of structural and binding data influence predictive performance (88)? How could one correct the impact of docking pose generation error on predictive performance and how does the implementation of web servers and stand-alone software make these tools freely available (89) (90) (91)?

One of the first appearance of machine learning in the field was in the role of an alternative to modelling assumptions in scoring functions. One of the first study of this kind was conducted by Deng *et al.* (92). They thought that non-parametric machine learning can be used to implicitly capture binding effects that are hard to model explicitly. In principle any possible kind of interaction can be directly inferred from experimental data by not imposing any particular functional form for the scoring function. In the previous mentioned study they used the distance-dependent interaction frequencies between a set of determined atom types, observed in two separate small datasets, as elements to model binding effects. This model was validated against several small external test sets (6 or 10 compounds). This study can be considered a valuable proof-of-concept that machine learning can produce useful scoring functions, besides of opening the way to the research on scoring functions based on machine learning. In the following years, support vector regression (SVR) was applied to produce family-specific scoring functions for five different ligand-protein systems using datasets with less than 100 complexes. The tests on the cross-validation data partitions produced excellent correlation coefficients. SVR was used also in combination with Inductive Logic Programming in order to obtain a set of quantitative rules that can be used in drug lead optimization for hypothesis generation (93).

In the very first years of study in the field of machine learning-based scoring functions, there has been much more research on machine learning approaches to Quantitative Structure–Activity Relationships (QSAR) bioactivity predictions. However, this type of

research is exclusively based on ligand molecule properties without taking into consideration the information from the protein structure. In fact, the model used is less complex with respect to a machine learning-based scoring functions. Indeed the results could be not reliable because the information on the protein is lacking.

Some early studies on machine learning scoring function (94) (95) used classic statistical approaches, such as linear models. The feature considered are hydrogen bonds, hydrophobicity, or van der Waals surface. The coefficient used in the training model are extracted from these latter features. Very soon it was clear that this simple approach cannot efficiently approximate free energies due to the nonflexible and simple nature of their linear modelling relationship (80).

The first scoring function based on machine learning to achieve high performance on a well-known benchmark was created by Ballester *et al.* (96) and is called RF-Score. The aim of their study is to create a function based on Random Forest (RF) (97) to predict ligand-protein binding affinity which can perform better than classic scoring functions. The descriptive model used for the ligand-protein complex is a structure based model. It consists in counting the number of a specific atomic pair, in which one atom belongs to the protein and the other atom belongs to the ligand. The atomic species considered by Ballester *et al.* (96) are C, N, O, F, P, S, Cl, Br, I. The space area in which the number of atomic couples considered are counted is a sphere of radius equal to 12 Å. Consequently, the total number of features which describe each ligand-protein complex is 81. Because of the composition of the proteins chosen for training and testing in the RF-Score model, 45 of the 81 features are composed by zeros. Therefore, each complex is characterized by a vector with 36 features. RF-Score is created using a Random Forest model with a number of trees equal to 500.

Wójcikowski *et al.* (98) further developed the idea proposed by Ballester *et al.* (96) creating a new scoring function, named RF-Score-VS. In this work the aim is to investigate the influence of including inactive molecules docked to targets in the training procedure. The analysis is conducted evaluating the screening power and the scoring power of the function and comparing them to the ones of classical scoring functions. The scoring power consists in simply predicting the ligand-protein affinity. The screening power consist in recognizing active ligands in a certain set. In fact, when a large database of compounds is screened, one then takes the best scored compounds for further evaluation. The screening power is measured with the enrichment test. The enrichment test tries to reproduce this screening operation counting how many active

compounds are among the best scored compounds. In particular the enrichment factor considered in this study is the top 1% (EF1%).

They use an analogue regression model with respect to Ballester *et al.* (96), based on random forest with the same number of trees. In this case the database used is composed by ligand-protein complexes created using different docking tools. They propose three different descriptive models trying to improve the structure based ligand-protein complex representation. The first model (v1) is exactly the same used by Ballester *et al.* (96). It uses a combination of ligand-protein atom-type pair counts on the binding site neighborhood in a single area of radius 12 Å. The atomic species considered are the same proposed by Ballester *et al.* (96). Because of this the total number of features considered for the ligand-protein complex description is 81. The RF-Score-VS v2 scoring function considers an additional feature for each complex: the Autodock Vina (56) partial score. In fact, this study used data coming from a docking procedure in the training and testing phase. The Autodock Vina (56) partial score is the docking score obtained by the particular docking software used in the data base creation. In this case the number of features for each complex is 82. The last version of scoring function proposed by Wójcikowski *et al* (98) is called RF-Score-VS v3. The descriptive model , instead of 1 interval of 12 Å, consider 6 intervals of 2 Å amplitude without considering the Autodock Vina partial score. In this case the number of features for each complex is 486.

More recently, deep learning has demonstrated the potential to exceed the capability of extracting information from the features used to describe such a complex situation, as the ligand-protein binding is. The flexibility of deep neural networks allows models, in principle, to learn successively higher orders of features from the simplest possible representations of the data at hand. In computer vision, for example, convolutional neural networks (99) applied to images can learn how to progressively detect edges, eyes, ears,... and finally faces, starting from early layers in the network, through intermediate network layers, to terminal layers of the network. While such advanced artificial neural network frameworks have led to immense advances in the fields of computer vision and natural language processing, they have only recently penetrated other areas, like the scoring function pipeline. In fact, the first machine learning scoring functions using convolutional neural network appeared not earlier than five years ago. This particular type of network and its structure, consisting in subsequent blocks with different functions, perform well with tri-dimensional databases, like the ligand-protein complex structure is. The aim of using convolutional neural network, with the tri-

dimensional complex structure as feature, is to learn more complex chemical features by optimizing both the model and featurization simultaneously.

One of the first studies in which a scoring function was created with a convolutional neural network, was performed by by Gomes *et al.* (100). They used two primitive convolutional operations: atom type convolution and radial pooling. The atom type convolution extracts features encoding local chemical environments from an input representation (Cartesian atomic coordinates) into a neighbor-listed distance matrix. Radial pooling consists in the dimensionality reduction of the output of the atom type convolution. This dimensionality reduction is done, both to prevent over-fitting through feature binning, and for reducing the number of parameters learned. Radial pooling takes as input the output by the atom type convolution.

Other important studies on machine learning scoring functions based on convolutional neural networks were published by Seo *et al.* (101) and Stepniewska *et al.* (102).

Seo *et al.* (101) proposed a scoring function called BAPA (Binding Affinity Prediction with Attention) using three kinds of neural network layers (convolutional, attention, and dense) for binding affinity prediction. Because machine learning scoring functions originally tend to have limitations, mainly resulting from a lack of sufficient interactions energy terms, the proposed model has two important features: a descriptor embeddings, that contains embedded information about the local structures of a ligand-protein complex, and an attention mechanism for highlighting important descriptors to binding affinity prediction, through the use of a weights vector.

Stepniewska *et al.* (102) have developed Pafnucy a scoring function based on a deep neural network. The model consists of two parts: the convolutional and the dense parts. The first uses three convolutional layers and the second three dense layers. In their approach, they consider the complex into a defined size of 20 Å cubic box focused at the geometric center of a ligand. Then the positions of heavy atoms are discretized using a 3D grid with 1 A° resolution. This approach allowed for the representation of the input as a 4D tensor in which each point is defined by Cartesian coordinates (the first 3 dimensions of the tensor) and a vector of features (the last dimension). In Pafnucy, 19 features were used to describe an atom: the atom types (species considered are B, C, N, O, P, S, Se, halogen and metal), hybridization, numbers of bonds with other heavy atoms, numbers of bonds with other hetero atoms, properties (hydrophobic, aromatic, acceptor, donor and ring), partial charge and molecules belonging (ligand or protein). According to the authors, this approach serves as a regularization technique as it forces the network to discover general properties of interactions between proteins and ligands.

Driven by deep learning-based approaches, which have rapidly emerged to provide state-of-the-art performances in different fields, and to find out how this class of models performs in molecular scoring tasks, also Jose Jimenez *et al.* (103) proposed an end-to-end framework, named KDEEP, based on 3D-convolutional neural networks for predicting ligand-protein absolute affinities.

They used a 3D voxel representation of both proteins and ligand using a van der Waals radius for each atom type, which in turns gets assigned to a particular property channel (hydrophobic, hydrogen-bond donor or acceptor, aromatic, positive or negative ionizable, metallic and total excluded volume), according to its rule. The contribution of each atom to each grid point depends on their Euclidean distance. The number of properties are duplicated to account for both protein and ligand, by using the same ones in each, up to a total of 16 different channels. These descriptors are computed on a fixed 24 $Å^3$ sub-grid centered on the geometric center of the ligand, in practice capturing a neighborhood of the binding site. The architecture network used by Jose Jimenez *et al.* (103) is adapted from the one proven successful in computer vision applications, such as SqueezeNet (104).

It is evident how strong and how recent the interest in applying machine learning to virtual screening and chemoinformatics is. Furthermore we are just focusing on a small piece of this wide field of study, such as ligand-protein binding affinity prediction, considering that we just focus on machine learning applied to ligand-protein binding affinity prediction.

3.2 Database

In the retrospective studies, the performance of scoring functions was evaluated on several public available benchmarking datasets. Here the principal databases used in the subject are presented, i.e. the Community Structure-Activity Resource (CSAR) (105), the PDBbind (106), the Directory of Useful Decoys (DUD) (107), and the Directory of Useful Decoys - Enhanced (DUD-E) (108).

Each study generated various classes of non-overlapping training and testing sets using the mentioned databases, intended to simulate possible application scenarios. In fact, it is important to consider that in any case, the complexes present in the training set are never present in the test set (or validation set, if used).

The reason for which shared datasets were created is facilitating the prediction of the binding affinities based on experimental complex structures. In fact, they are composed of 3D complex structures coming from experiments. The availability of experimental ligand-protein complex structures allows the structure-based featurization to correlate with the ligand-protein binding affinities with the precise binding interactions.

The most important database in docking and virtual screening pipeline is the PDBbind database, which has already been presented.

The CSAR database disseminated experimental datasets of crystal structures and binding affinities for diverse ligand-protein complexes. The acronym stands for Community Structure-Activity Resource. The repository is hosted by University of Michigan. Some data were generated in house at the University of Michigan, while others were collected from the literature or deposited by academic labs, national centers, and the pharmaceutical industry. As anticipated, also for this project the original aim was to create better datasets to train scoring functions and develop new docking algorithms. The DUD and DUD-E datasets are composed of data created by docking tools. DUD stands for directory of useful decoys. They were originally designed to assess docking enrichment performance by distinguishing the actives ligands among a large database of computationally generated non-binding decoy molecules.

DUD is designed to help test docking algorithms by providing challenging decoys. It contains a total of 2950 active compounds against a total of 40 targets. For each active, 36 "decoys" with similar physical properties, but dissimilar topology, are contained. DUD is provided by the Irwin and Shoichet labs in the Department of Pharmaceutical Chemistry at the University of California, San Francisco (UCSF) (109).

DUD-E is an enhanced and rebuilt version of DUD. It contains 22886 active compounds and their affinities against 102 targets, an average of 224 ligands per target. In addition, it contains 50 decoys for each active having similar physico-chemical properties but dissimilar 2-D topology.

In one of the first study on the topic of machine learning scoring function, Ballester *et al.* (96) used the 2007 PDBbind release. In order to generate a refined set suitable for validating scoring functions, the authors of the study applied a series of conditions for the data. Only complex structures with a resolution of 2.5 Å or better were considered. Only complexes with known dissociation constants (K_d) or inhibition constants (K_i) were considered. They left those complexes with assay-dependent IC_{50} measurements out of the refined set. Still there is no unique affinity value considered as univoque ligand-protein affinity. In addition, because not all molecular modelling software can handle

ligands with uncommon elements, only complexes with ligand molecules containing just the common heavy atoms (C, N, O, F, P, S, Cl, Br, I) were considered. This process led to a refined set of 1300 ligand-protein complexes with their corresponding binding affinities. The predictions of scoring functions were tested on the core set, which comprised 195 diverse complexes with measured binding affinities.

In the study by Wójcikowski *et al.* (98) the database was composed of the complexes deriving from the combination of 102 protein targets with a group of active molecules for each target (224 ligands on average) and decoys (50 decoys per active ligand). The decoys were obtained from the DUD-E (108). The total database is composed by 92750946 data. The structure of the complexes was generated with three docking programs: AutoDock Vina (the Smina implementation, [http://smina.sf.net/]) (56) (110), Dock 3.6 (111), and Dock 6.6 (112) (113). Only one best scoring ligand pose according to the docking tools was chosen for the scoring function model building.

Gomes *et al.* (100) used two subsets of the PDBBind 2015 dataset: core (195 structures) and refined (3706 structures). The crystal structures present in the core and refined datasets were obtained at a higher resolution and cleaned more thoroughly than the full dataset, considering more stringent requirements on the quality of the complex structure, quality of the binding data, and the nature of the complex. They used the two databases separately. All train/test splits follow the 80/20 ratio.

Seo *et al.* (101) also used PDBbind database, but in version 2016 and 2018, for training the model. The training set is composed of 3689 complexes. They mainly used, as test dataset for model performance evaluation, CASF-2016 dataset (285 complexes) and CASF-2013 dataset (195 complexes). The corresponding binding affinities are expressed with pK_a, which can be defined using $-\log(K_d)$ or $-\log(K_i)$.

Also Stepniewska *et al.* (102) used PDBbind 2016 as principal database. The general and refined sets were used to train the model and to select the hyperparameters (11906 data). While the core set (290), CASF-2013 (129), and Astex Diverse Set (114) database was used as an external test set. The Astex database is an independent source with respect to PDBbind project. Like for Seo *et al.* (101), their corresponding binding affinities is expressed with pK_a, which can be $-\log(K_d)$ or $-\log(K_i)$.

For training the model, commonly to previous studies, Jose Jimenez *et al.* [26] used the refined set of the PDBbind database, without including the complexes belonging to the core subset of the same database, for a total number of 3767 complexes. This latter subset is used as the principal test set. It is composed of 290 complexes. In addition, they perform secondary tests on other databases like CSAR NRC-HiQ set 1 and set 2,

CSAR 2012 and 2014. As in the previous mentioned studies, they do not use an univoque measure for the binding affinity. In fact, there is no distinction between K_d and K_i, that is dissociation and inhibition constants. They consider pK_a, which can be either $-\log(K_d)$ or $-\log(K_i)$.

3.3 Test types

Several studies underlined how machine-learning scoring functions have outperformed classical ones in binding affinity prediction. More recent studies, besides comparing their machine learning performances with classic scoring functions ones, try to outperform the already existing machine learning scoring function, for a sort of performance escalation.

In order to determine the scoring functions performance, a series of different tests were performed. In this paragraph, the principal types of tests used to describe a machine learning scoring functions performance and the corresponding results in the most important studies on the subject are described.

Basically, there are two main type of tests: vertical test and horizontal test. Normally, a database is composed of complexes coming from a relative circumscripted number of proteins. Each protein is bounded with many ligands. It is possible to consider the complexes coming from the same protein as a subgroup inside the database. In this case, the entire database is represented by an ensemble of subgroups representing the complexes of each protein, as schematically shown in the Figure 3.1.

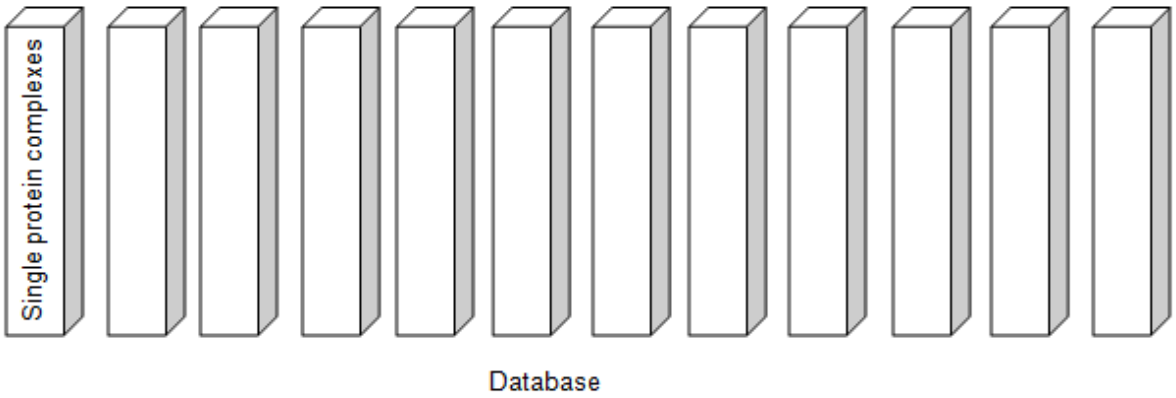

Figure 3.1 Schematical representation of the database

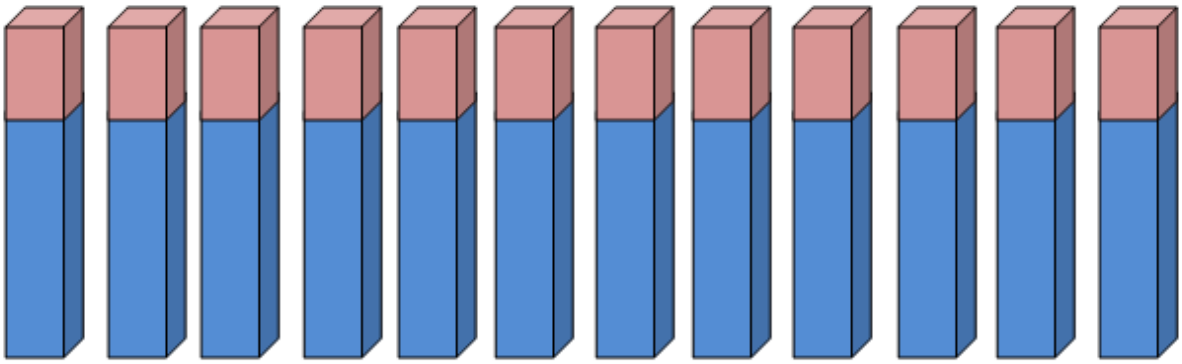

Figure 3.2 Schematical representation of the database subdivision in the horizontal test

A horizontal test consists in subdividing the database, represented in the way of Figure 3.2, horizontally into the training and test set, according to a determined percentage. In this explanation we just consider a subdivision of the database in these two groups, but in the case the validation set is used, another subset is created for this latter set. The subdivision is still horizontal. Each subgroup, represented by the complexes deriving from a single protein, is divided into complexes belonging to the training and test set. For this reason we can find complexes coming from the same protein in both the training and test subsets. The scheme in Figure 3.2 is not representing exactly the database because the subset of complexes coming from each protein could have different sizes. In addition, the subdivision is not equally broken down among the subgroups. In fact, it is done considering the entire database. For this reason it can happen that the complexes of a particular protein are included for the 85% in the training set and for the 15% in the test set and the complexes of another protein are subdivided according to the percentage 75-25. Anyway, Figure 3.2 shows the average subdivision of the database into training and test set for a horizontal test.

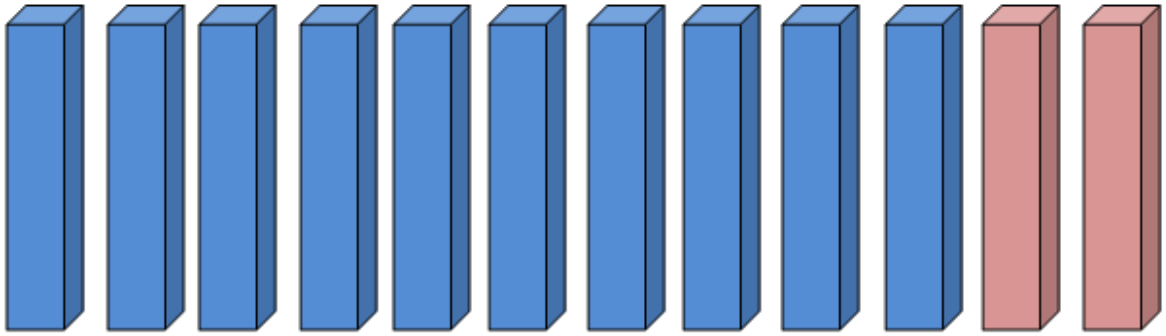

Figure 3.3 Schematical representation of the database subdivision in the vertical test

The vertical test is schematically represented in Figure 3.3. It consists in subdividing the database as represented in Figure 3.1, vertically. In this case, the training set is composed of complexes of proteins not present in the test set and vice versa. Because of this, when you use a vertical test, you build your model using some proteins and then you test it on proteins never seen by the model.

As anticipated, the tests are done to compare the performance of the scoring functions in pratical applications. The vertical test can describe a real-case scenario. In chemical and pharmacological laboratories, scientists often aim at discovering new drugs or studying the effects of some active principles on a certain protein. A typical situation is that the affinity between some ligands and a protein (never addressed before) must be predicted. The usual scientist's instruments are the scoring functions, classical or based on machine learning. In addition, in his/her domain, he/she has a number of data, experimental or synthetic, of which the affinity and a precise or less precise molecular structure are known. These data could be the ones used to train the machine learning scoring function. But, probably, any data on the protein under study are known because the process is acted to discover something new. This could describe the case in which the world is entangled now. That is the Corona virus epidemic. A vertical test can well reproduce this scenario. In fact, in a vertical test, the network is trained on complexes deriving from some specific proteins. The consequent test is performed on a pool of data which are not deriving from any protein present in the train set. In the present case, the test is called vertical test.

A variation of the vertical test is the per-target vertical test, schematically shown in Figure 3.4.

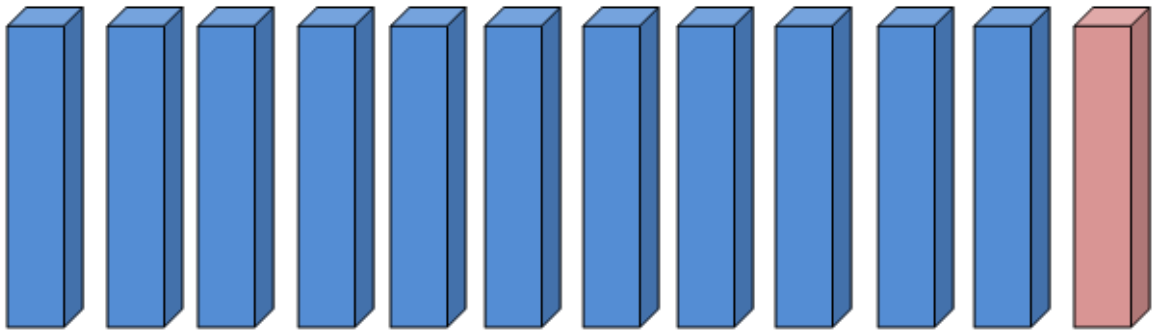

Figure 3.4 Schematical representation of the database subdivision in the per-target vertical test

The scoring function is trained on the entire database except on the complexes deriving from one protein. These are used as test set. Using this type of test the typical scenario of medicine discovery, where the target is previously selected but never studied before, is described. Following the example of the Corona virus, this test could describe the case where you have selected the Spike protein to be the target protein of your study and the research of effective ligands begins.

On the other hand, a horizontal test does not describe the situation well, because the machine learning scoring function is tested on complexes similar to others already seen. In particular, the protein in your test is already experimentally measured in the reactions with many other ligands. This is in contrast with the situation described above. This approach can mimics experiments where docking is performed on targets for which there are already known active ligands and virtual screening is done to find new ones. It is a real, but less interesting scenario.

Ballester *et al.* (96) created the first machine learning scoring function to achieve high performance on a well-known benchmark. That performance was measured using a standard horizontal test. The performance parameter taken in consideration is the Pearson Correlation Coefficient, R_p. A value of R_p= 0.78 was obtained. The question of how much of the predictive ability of machine learning scoring function is due to learning the true relationship between the atomic-level description of structures and their binding affinities is open since this pioneristic study. The authors tried to answer this question mixing the database. In particular they assigned a wrong affinity value to the ligand-protein complexes present in the test set. The aim is to verify if the RF-Score is able to

reach similar level of R_p using a correct or incorrect test set. If the R_p reached is the same in both cases, probably the machine learning scoring function is not learning the binding properties of the complexes. Otherwise, probably, it is learning the binding properties of the complex. The mean R_p obtained with this test has an absolute value of 0.18, which is considered a proof of the correct functionality of RF-score.

Wójcikowski *et al.* (98) made a precise distinction in the tests they performed. When the screening power is evaluated, they perform three types of test for each of the three scoring functions: horizontal, vertical, and per target vertical test. The results in horizontal test show a dramatic increase of EF1% performance between the best classic scoring functions compared to machine-learning scoring functions: around two to, even, 15 times increase depending on the docking engine and scoring function. The vertical test shows that there is a drop in EF1% performance with respect to the horizontal one. Nevertheless, this result is still better than the one obtained from a classical approach, even if the difference is very small. Eventually, they trained a separate SF for each of the DUD-E targets (per-target scoring functions) tested on per-target test in order to verify if tailored functions perform significantly better than a generic function. Most of the per-target functions evaluated in the Wójcikowski *et al.* (98) study tend to perform only slightly better than the generic, unique, function (trained on all available data). This slight improvement can be observed in particular if the data on the target chosen are numerous. Because the enrichment test do not show if these top 1% molecules are actually the most active ones, Wójcikowski *et al.* (98) check if machine-learning methodology predicts binding affinity better than a classical SF, using version 2 (v2) of RF-score VS. The test is conducted measuring the R_p in the horizontal and vertical tests and obtaining respectively a value of $R_p = 0.56$ and $R_p = 0.20$. The test confirm the difficulty of the actual machine learning scoring function in vertical test.

Gomes *et al.* (100) consider four methods of splitting PDBBind core and refined sets into subsets for train/test evaluations. These are employed for four different types of tests. The tests are called random, stratified, scaffold, and temporal. The random split consist in a usual horizontal test. In fact, they randomly split samples into train/test subsets. The stratified split sorts examples in order of increasing inhibition constant K_i, and then it chooses 10 samples at a time and randomly splits these samples into train/test subsets to ensure that each set contains the full range of inhibition constant present in the parent dataset. It can be considered a horizontal test because the same protein can be present both in train and test set. Scaffold splitting considers ligand

molecules scaffold (the structure obtained removing side chain atoms). Common scaffolds are placed in the train set and uncommon scaffolds are placed in the test set. This split attempts to separate structurally distinct molecules into train and test sets. For this reason it represents a kind of vertical test, even if the ligand structure is considered and the separation criteria are based on common and uncommon structures. Temporal splitting was performed based on the year that the ligand-protein complex was entered in the Protein Data Bank. This split tests the ability of the learning algorithm to use prior historical data to predict results of future experiments, similar to typical use in prospective drug discovery. Based on the definition reported at the beginning of this paragraph, this is an horizontal test too. To determine the train and test set performance, the squared Pearson Correlation Coefficient (R_p^2 of $\log K_i$ were evaluated). The results obtained in the 4 types of test are the following. In the random test, the scores R_p^2 = 0.448 and R_p^2 = 0.508 are obtained, respectively, using the PDBbind core set and the PDBbind refined set. In the stratified test the scores R_p^2= 0.116 and R_p^2 = 0.491 are obtained, respectively, using the PDBbind core set and the PDBbind refined set. In the scaffold test the scores R_p^2= 0.043 and R_p^2= 0.267 are obtained, respectively, using the PDBbind core set and the PDBbind refined set. In the temporal test the scores R_p^2 = 0.251 and R_p^2 = 0.529 are obtained, respectively, using the PDBbind core set and the PDBbind refined set for training and testing the model. It is possible to observe that the performance obtained in the horizontal test are sensibly higher than the one obtained in the scaffold test, which can be considered as a vertical test, even if facilitated. In addition, the performance of Gomes *et al.* (100) scoring function (ACNN) is compared with other machine learning scoring functions based on models already proposed in literature, also much simpler with respect to the one proposed by Gomes *et al.* (100). The performance obtained are comparable.

Seo *et al.* (101) evaluated the performance of binding affinity prediction models via different metrics. In particular, we point the reader's attention to the Pearson's Correlation Coefficient (R_p). The conducted test is a horizontal test. In the main test sets chosen, that are based on the CASF-2013 and CASF-2016 databases, the scores obatined are respectively, R_p = 0.77 and R_p = 0.82. In addition, they perform a kind of vertical test to evaluate the performance of the model according to protein structure similarity or ligand structure similarity. The structure similarity is evaluated based on Li *et al.* (115) study. It is not possible to define these test as a proper vertical test, because the criterion of avoiding same protein, both in training and test set, is not respected.

Nevertheless, in this latter test there is a flexion of the BAPA (101) performance, with respect to the horizontal test, of less than 0.1, considering the R_p parameter. Eventually, they compare BAPA with four existing popular prediction models including RF-Score v3 (79), Pafnucy (102), PLEC-linear (116), and Onionnet (117). All the just mentioned models are trained and tested using the same datasets of this study and the results are comparable.

In Stepniewska *et al.* (102) the correlation between the scores and experimentally measured binding constants is assessed mainly with the Pearson's correlation coefficient R_p. They perform horizontal tests in order to measure the performance of Pafnucy scoring function using the PDBbind core set, CASF-2013, and Astex database. They obtain respectively R_p = 0.78, R_p = 0.70, and R_p = 0.57. The performance of Pafnucy scoring function are similar to the ones of other machine learning scoring functions in horizontal test, as already presented.

As in previous studies, Jose Jimenez *et al.* (103) use the usual horizontal test and measure the scoring function performances with Pearson's correlation coefficient R_p as the main performance parameter. KDEEP shows better performance in the horizontal test if the PDBBind core set is used (R_p = 0.82). Instead, if other databases are considered, the performance is reduced (R_p = 0.72 and 0.65 respectively on CSAR NRC-HiQ set 1 and CSAR NRC-HiQ set 2, and R_p = 0.37 and 0.61 respectively on CSAR 2012 and CSAR 2014). Also in this case the mean KDEEP performances are comparable with the ones of other mentioned machine learning scoring functions on horizontal test. Besides of comparing the scoring function performances to other machine learning scoring functions, in our study a comparison is also made with basic scoring function, like molecular weights scoring function. This latter is a scoring function which simply attribute a ligand-protein binding affinity assuming a linear dependence as a function of the molecular weights of the molecule. KDEEP highlight clear better performance in the horizontal test with respect to a molecular weights scoring function. Eventually, they perform a secondary test in which KDEEP is tested on a set of data composed only of complexes of one protein, not present in the train set, that is, a per-protein vertical test. In this case, the performances are significantly lower than the ones of the horizontal test and do not overtake the performance of classic scoring function, on the contrary, they are comparable. The weighted average R_p measured by Jose Jimenez *et al.* (103) in the described case is R_p = 0.34.

Basically, all machine learning scoring functions created in recent years overperform the classic scoring functions. In fact, as discussed, most of the studies compare the results obtained by machine learning scoring functions with respect to classic scoring functions in a particular test, obtaining a positive feedback. In Table 3.1, the performances of the most important machine learning scoring functions introduced in this chapter are reported. The aim of the Table 3.1 is to furnish a clear description of the performance state of the art of this type of scoring function and to compare each other based on the same performance parameter and class of test. In fact, as presented, different research groups can use different type of performance descriptors. In this situation, it is difficult to have a clear picture of the state of the art. In Table 3.1 the Pearson correlation coefficient is used to compare all the scoring functions because every research use it in addition to other different parameters. The scoring functions are listed according to their authors. The test considered are the horizontal, vertical, and per-target vertical test. The definition of these tests has been reported previously in the present paragraph. An additional column is added to record the results obtained in tests similar to the vertical test. These tests are called " Vertical test "Kind of"". In this category the tests that avoid the contemporary presence in training and test set of similar complexes, for any type of criteria (protein sequence-based similarity, ligand scaffold similarity,...), are reported. These tests are not exactly vertical test according to definition reported, but they try to reproduce similar conditions of vertical tests. In the table, between round brackets the test set used in a certain test is reported. Always in the table, between squared brackets, the name of the test performed according to the author of the research is reported.

The best performing benchmark for machine learning scoring function is, for sure, the horizontal test. In fact, it is the test in which the best performances are recorded. However, we observe that machine learning scoring function encounter some difficulties in vertical tests, or in similar benchmarks. The teams that perform vertical tests always recorded a general decrease of the machine learning scoring function performances with respect to what was measured in the horizontal test. If the test is strictly vertical, the performance decrease is sensible, but also in the cases where a kind of vertical test is adopted, a performance decrease is already present.

On the other hand, vertical and horizontal test for classic scoring functions have no meaning because they do not sustain a training procedure. When machine learning and classic scoring functions are compared in any kind of test, it is the testing pool to be the

same. In fact, the performance of classic scoring functions are stable between a horizontal and vertical test. The gap that appeared between these latter and machine learning scoring functions in horizontal tests, usually disappear in vertical test.

Study	Performance: R_p(database used)[Test type used in original study]			
	Horizontal test	Vertical test	Per-target vertical test	Vertical test "Kind of"
Ballester *et al.*	0.78(PDB core)	/	/	/
Wójcikowski *et al.*	0.56(DUD-E)	0.20(DUD-E)	/	/
Gomes *et al.*	0.67(PDB core); 0.71(PDB ref.) [Random]. 0.34(PDB core) 0.70(PDB ref.) [Stratified]. 0.50(PDB core) 0.73(PDB ref.) [Temporal].	/	/	0.21(PDB core) 0.52(PDB ref.) [Scaffold]
Seo *et al.*	0.77(CASF-2013) 0.82(CASF-2016)	/	/	0.74(CASF-2013)
Stepniewska *et al.*	0.78(PDBcore), 0.70(CASF-2013) 0.57(Astex)	/	/	/
Jose Jimenez *et al.*	0.82(PDBcore) 0.37(CSAR 2012) 0.61(CSAR 2014)	/	0.34(PDBcore)	/
Jincai Yang *et al.*	0.84(PDB ref.) 0.73(PDB gen.) 0.71(PDB ref.) 0.60(PDB gen.)	/	/	0.63(PDB ref.) 0.54(PDB gen.) [sequence-based splitting]. 0.48(PDB ref.) 0.42(PDB gen.) [ligand scaffold similarity]

Table 3.1 Comparison among most important machine learning scoring functions performances

3.4 Performance doping factor

Of course, machine learning scoring functions can be an innovative solution in the field of virtual screening and chemoinformatics. However, not all the researchers agree on the incredible performances of these functions. In this paragraph we deal with some possible bias that influences the performances presented above.

Jincai Yang *et al.* (118) go into details of Josph Gomes *et al.* (100) study and their scoring function, called ACNN, to demonstrate that often machine learning scoring function performances are doped by the chosen database and testing method. In their study, they demonstrate that same levels of performance are obtained considering the whole complex structure or only the protein or the ligand structure singularly. ACNN models did not require learning the essential ligand-protein interactions in complex structures and achieved similar performance even on datasets containing only ligand structures or only protein structures. This means that machine learning scoring functions simply recognize similar proteins or ligands, but do not learn the propriety of the binding mechanism. In addition, the authors performed data splitting based on similarity clustering (protein sequence or ligand scaffold), significantly reduced the model performance. In fact, already Li *et al.* reported that the protein similarity impacts the performance of artificial intelligence models (115). They obtained a general reduction of 0.2 in the R_p^2 both if the clustering is based on protein sequence or ligand scaffold with respect to a classic horizontal test. Jincai Yang *et al.* (118) concluded that biases are widely present in the two database under study, PDBbind and DUD-E, and the performance of scoring functions using these database arguably suffers from the data redundancy caused by the protein and ligand similarity. For this reason they propose to use sufficiently large and unbiased datasets for training robust artificial intelligence models to accurately predict ligand-protein interactions. This implies verifying that the database does not present redundancy of similar structures in train and test set since this can lead to overestimate the scoring function performance. It is easy to understand that, according to Jincai Yang *et al.* (118), the horizontal test identifies the conditions that should be avoided because it can overestimate the performance of the function.

4 MLP Scoring function

As anticipated in the Introduction, the objective of this thesis is pursued through the creation of a new machine learning scoring function. As regression model, it is used a multilayer perceptron. For this reason, the scoring function is called MLP scoring function. In this chapter, we start presenting the creation of ligand-protein complex descriptive model and the features matrix and target vector. Then, we analyze the regression model.

4.1 Ligand-protein complex descriptive model

The machine learning scoring function faces the problem of complex description, with particular reference to the binding zone, in terms of a matrix. As already discussed, in previous studies, Pedro j. Ballester *et al.* (96) proposed the model where it counts the number of a certain atomic couples in the area of the binding pocket. Woljacowsky *et al.* (98) considered the same model proposed by Pedro j. Ballester *et al.* (96). In addition to this model, they proposed two more versions of the descriptive model. Other studies in the field of machine learning scoring functions use convolutional neural networks. For the particular configuration of this networks, a tri-dimensional description of the complex is needed. In our case, the regression model chosen is a multilayer perceptron, and a bi-dimensional descriptive matrix is used as input.

The archetype proposed by Ballester *et al.* (96) and then Wójcikowski *et al.* (98) is resumed in the present research. This model involves counting the number of particular atomic couples in a certain range of distance around the pocket. The atomic couple is composed by an atom belonging to the protein and an atom belonging to the ligand. The list of protein atomic species considered is reduced with respect to the previous studies just because some of the atoms considered there are never present in complexes we considered. However we consider the hydrogen among the atomic species taken into consideration. The atoms considered for the ligand and for the protein are reported hereafter, respectively:

[H, C, N, O, F, P, S, Cl, Br, I];

[H, C, N, O, P, S].

All the possible couples created combining the atoms belonging to the ligand list with the ones to the protein list are considered.

Consequently, the number of different possible couples of atomic species is 60.

The atoms belonging to the ligand list are all the possible atoms present in the ligand except for metals, because it is unusual to find ligands with metals. In fact in the databases used in this study, ligands with metals are not present. Therefore, if metals had been included in the ligand list, the result would be a training matrix with larger size in which the columns dedicated to the couples with metals are filled by zeros.

In particular, we are inspired by the Wójcikowski *et al.* (98) descriptive model because we consider more than one interval in which the number of atomic couples are counted, as in version 2 of their RF-Score VS. A study is conducted on what the best descriptive model is in terms of interval amplitude and radius area of the pocket (Figure 4.1). The different amplitude considered are 1.5 Å, 2 Å, and 3 Å. It means that the counting of the number of couples happens in each interval of the specified amplitude until the pocket area is covered. The network is trained using these different descriptive models, growing the area radius from time to time. A scoring function is created using each different descriptive model. The regression model used is the multilayer perceptron trained with experimental data. The score used to evaluate the best descriptive model is R_p . The type of regression model used and the reasons why we are using it are discussed later in this chapter. The results show that the best descriptive model in terms of amplitude is 2 Å, as in the model employed by Wójcikowski *et al.* (98). The effective binding information seem to be in the area of 6-8 Å. In fact, every interval amplitude used in the description model produces the best performance in terms of R_p , when an area of radius equal to 6-8 Å is considered . In our study a maximum distance of 8 Å is considered.

Four ranges of intermediate distances are considered between 0 and 8 Å, in particular, 0-2 Å, 2-4 Å, 4-6 Å, and 6-8 Å. The compiling of the matrix is made calculating the absolute distance between the element of the ligand and the element of the protein for all the considered ligand-protein couples. The distance is simply evaluated using the Cartesian coordinates of the two atoms. Once the distance is calculated, the counter of a particular couple and of the intermediate range in which the distance is included increases by one.

The matrix used to train the neural network is composed by a number of rows equal to the ligand-protein complexes considered and a number of columns equal to 240. The number of columns is the product between the considered couples and the distance

ranges for each couple. As an example, the first column of the matrix contains the number of couple H-H in the distance range 0-2 Å, the second one contains the number of couple H-H in the distance range 2-4 Å and so on until the seventh column which contains the number of couple H-C in the distance range 0-2 Å.

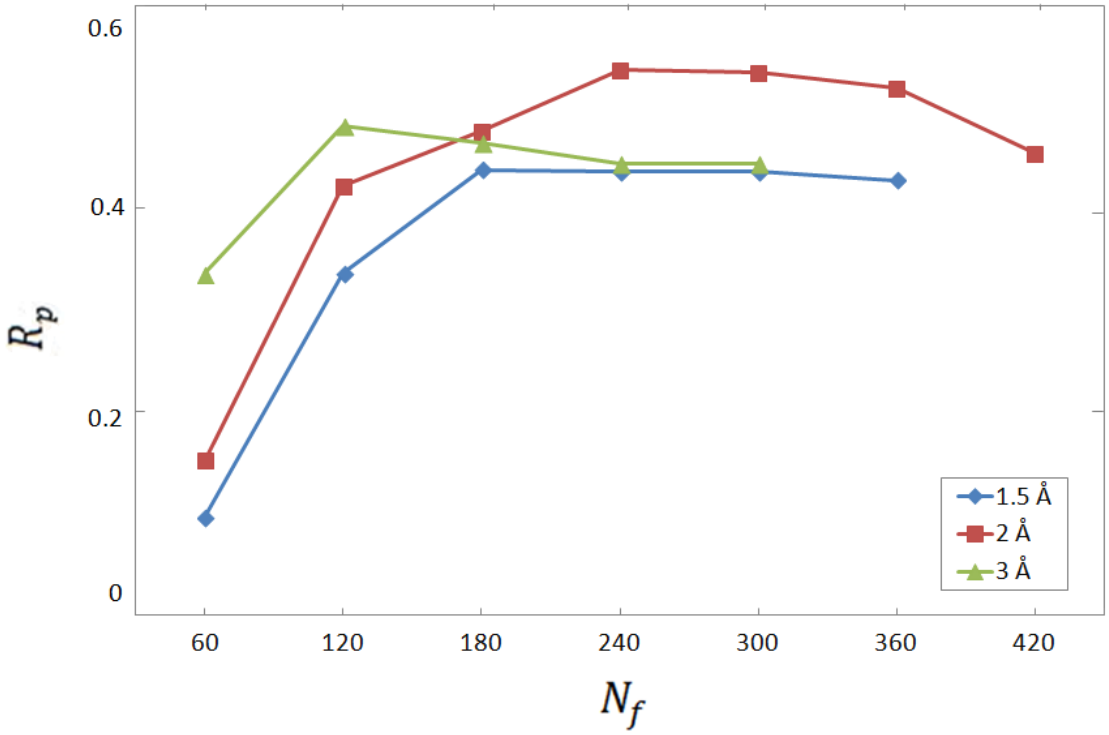

Figure 4.1 Ligand-protein descriptive models comparison

The target matrix is simply a vector in which the target value considered (ligand-protein affinity or docking score) is reported. Each line of the training matrix and the relative component of the vector are the data used to describe a particular ligand-protein complex in this study.

4.1.1 Database normalization

The database is composed of a training matrix and a target vector. The components of the training matrix can assume values between 0 and numbers of the order of 10^5. The target vector can have different distributions according to which database or target value is considered. In this paragraph, we treat the process performed to standardize the database, both for training matrix and target vector.

In the training matrix the presence of large values among the components of the matrix can compromise the learning process. For this reason a normalization operation is needed. Different types of normalization can be applied to the training matrix:

A. No normalization.

B. Normalization using overall maximum.
   Each component is subdivided by the maximum value present in the matrix.

C. Normalization using $\log(feature\ value)$.
   The component of the training matrix is done by the logarithm of the original component.

D. Normalization using the average value of each column.
   Each component is divided by the mean value of the column to which the component belongs.

E. Normalization using the maximum of each column.
   Each component is divided by the maximum value of the column to which the component belongs.

A test is conducted to verify what the best normalization is for the model built in this study. In order to do this, as for choosing the best description model, a scoring function is created using each normalization model. The regression model used is the multilayer perceptron trained with synthetic data. The score used to evaluate the best descriptive model is R_p . The type of regression model used and the reasons why using it are presented later in this chapter. The results of the test are shown in Figure 4.2. The normalization model B is applied in the rest of this study.

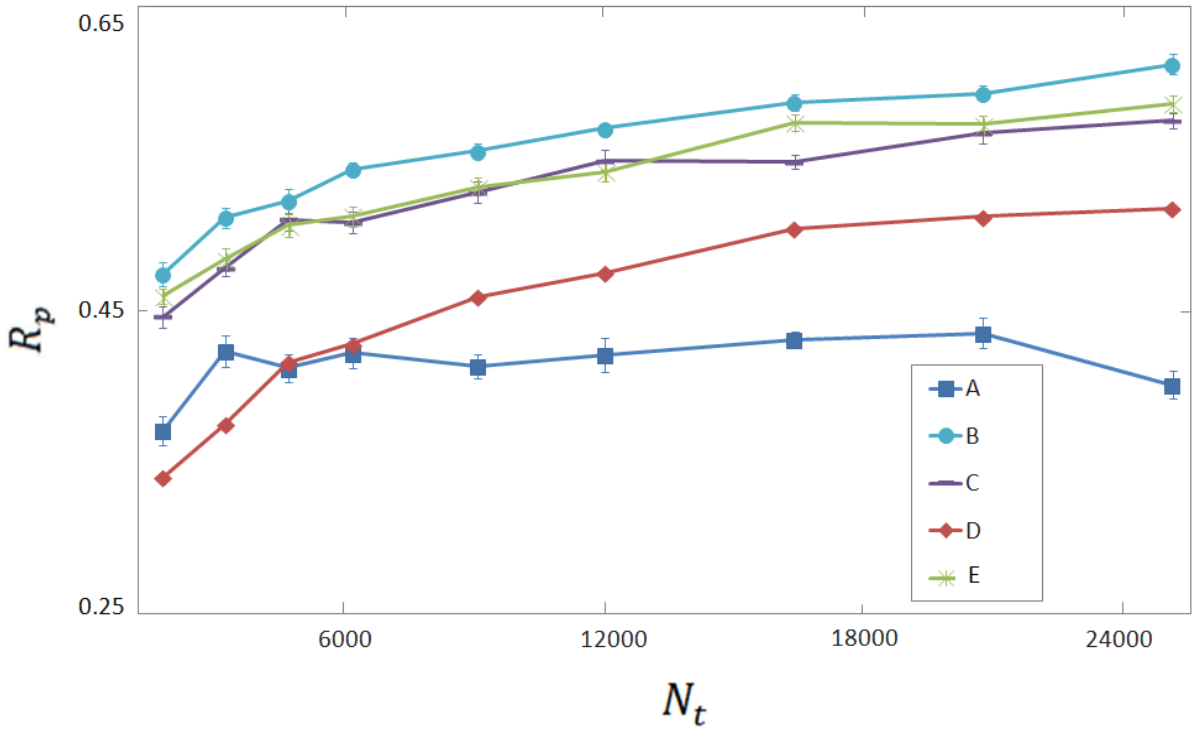

Figure 4.2. Features normalization method

In the target vector, we want a normalization procedure which guarantees a similar distribution for each type of data and target value. In particular we want distributions of target data with a mean value of 0 and a standard deviation equal to 1. In this way the training of a same neural network with two different sets of data, as done when the transfer learning technique is used, is homogeneous and produces reliable results. Hereafter, the operation used to normalize the target values is reported:

$$d' = \frac{d - \mu}{\sigma}$$

where d' is the normalized target value, d is the original target value, μ is the mean value in the considered database and σ is the standard deviation in the same database. The result obtained is a database where the target data distribution has mean value equal to zero and standard deviation equal to 1.

4.1.2 Database construction: code analysis

Following the previous qualitative description, now we deep on the training matrix and target vector creation process starting from the .pdb files.

All the procedures are implemented with Python.

Python is a "high-level" object-oriented programming language suitable, among other uses, for developing distributed applications, scripting, numerical computing and system testing. It was designed by Guido van Rossum in the early nineties. It is a multi-paradigm language that has among its main objectives: dynamism, simplicity and flexibility. It supports the object oriented paradigm, structured programming, and many functional and reflection programming features. It is often studied among the first languages for its similarity to a pseudo-code. It is frequently used to reproduce the creation of software thanks to the experimentation flexibility, which allows the programmer to organize ideas during development (119).

TRAINING MATRIX

Considering the .pdb files subdivided in various folders inside the same upper folder, the first step is to open sequentially each folder and, then, the files inside the folder. The environment *os* and the command *os.listdir()* allow to open the upper folder and to have a list of the files contained in each folder. In order to proceed in the process a series of ad hoc functions are created. The following paragraphs describe each function. Subsequently, we will describe how to apply the functions to create the training matrix.

Function "*openf()*".
Input: name of file .pdb.
Output: selection of the text contained in the .pdb file formatted as a list in which each item is a text line.
The function puts the input in an address in order to open that .pdb file. Once the file is opened in reading mode, the function *readlines()* transforms the file in a list in which each line of the file is an item of the list. Subsequently, the *openf()* function determines the beginning and the end of the part of the file which contains interesting information for the research. In order to achieve this, it looks for the word "ATOM" in the characters 0:4 of each line and the word "CONECT" in the characters 0:6 of each line. The word "ATOM" at the beginning of the line means that the list with the information about each element of the complex ligand-protein is started. The word "CONECT" is used in the last part of the .pdb file to describe the various connections present in that molecule. This kind of information is useless in the research conducted in this thesis, it means that the part of text containing useful information for the research is over. The function *openf()* selects the index of the first line containing "ATOM" and the index of the first line containing "CONECT". It selects the text between these 2 index, and it returns it.

Function “*DistanzaDaTuttiHETATM()*”.

Input: output of function *openf()*

Output: List of three elements vector composed by all possible ligand-protein couples in which the first element is the atom of the ligand, the second one is the atom of the protein, and the third one is the Cartesian distance between the first two.

The function “*DistanzaDaTuttiHETATM()*” creates a list with all the possible ligand-protein couple and the associated Cartesian distance. Each element of the list is a vector with three components: the atom of the ligand, the atom of the protein and the distance between the two. The function reads all the lines of the input list with a *for* cycle. Because the ligand-protein couples are considered in this order, the code first selects the element belonging to the ligand. It does not consider the water molecule present in the ligand because in the .pdb tracking mode they are assigned to the ligand, but they are not real part of this. They are only necessary to complete the connections between the protein and the ligand. When the *if* logic, used to find elements of the ligand, is verified, the line of the considered text is subdivided using the command *.split(). E*ach element of the line separated by a space character from the next one becomes an element of a list. The list is composed by all the elements of the line. For each ligand components, another *for* cycle and a *if* logic is used to determine all the possible couples between that element of the ligand and all the elements of the protein. With the *for* cycle all the lines of the starting list are scrolled. The *if* logic considers only the elements belonging to the protein. When this last *if* logic is verified, one element is added to the output list. The element is composed by a vector of three elements, the first is the atomic specie of the considered ligand atom, the second is the atomic species of the considered protein atom and the third is the distance between these two elements. The distance is calculated using the *math* environment and the *math.sqrt()* function. This function calculates the square root among the sum of the squared difference of the ligand-protein relative coordinates.

During the study of the function *DistanzaDaTuttiHETATM()*, some small bugs were discovered in the .pdb files. The Cartesian coordinates of the atoms can be composed by 8 digits (the minus sign, 3 digits, the comma and 3 decimals, encountered in order from right to left in the .pdb file). When all the digits are used in y and z coordinates, no space is supposed to be present between these and the previous coordinates. For this reason, the function cannot recognize that two coordinates are present instead of one, and some unexpected crashes are encountered while the program is running. In order

to solve this problem, a substitution is made when the combination “-1” is encountered. The substitution is made adding a space character before the combination “-1” and so “ -1”. In fact, the hundreds digit in the Cartesian coordinates never takes a value different from 1. The same problem is encountered in the columns which precede the Cartesian coordinates in the .pdb files. These columns contain information about the type of amino-acid in which the considered atom is included, the type of molecule in which the atom is include and some other information useless for the research conducted in this study. For these reasons, it is chosen to cut a part of each line, until the 28th character. The 28th character of each line of the .pdb file coincides with the beginning of the coordinates for each atom.

Function “*Statisticizza()*”.

Input: output of function *DistanzaDaTuttiHETATM()*

Output: Dictionary containing as keys all the considered ligand-protein couples and as key-values an numpy array composed of 6 elements corresponding to the considered distance ranges. Each element of the array indicates how many couples of the key couple are present in that range.

The function *Statisticizza()* creates a dictionary starting from the output list of function *DistanzaDaTuttiHETATM().* As input value, in addition to the output of function *DistanzaDaTuttiHETATM()*, it takes two more lists, one containing all the ligand-protein couple taken into account and the other containing the range of considered distance. For each component of the ligand-protein couple list, *Statisticizza()* creates a numpy vector composed of 6 zeros and scrolls all the lines of the output list of function *DistanzaDaTuttiHETATM()* looking for the corresponding couples. Once the considered couple and the couple of the list coincide, the function compares the Cartesian distance with the distance range. Using an *If* cycle it adds 1 to the numpy vector in the position of the range in which the Cartesian distance is included. The smallest distance range is the element with index 0 of the numpy vector and the largest one is the element with index 5. When all the lines of the output list of function *DistanzaDaTuttiHETATM()* are read, *Statisticizza()* writes the numpy vector in the dictionary using as dictionary key, the atomic symbol of the two element composing the couple.

Function “*ModifMatrice()*”.

Input: output of function *Statisticizza(),* numpy matrix with dimensions $L\ x\ D$, (L is the number of .pdb files. D is the number of considered ligand-protein couples multiplied by the range of considered distances); matrix line index in which modifying the matrix.
Output: /
ModifMatrice takes as input a matrix (composed by zeros) and substitutes each line with information deriving from the output of function *Statisticizza.* It considers the matrix line index passed through the input to the function. In that line it substitutes to the original line, the numpy arrays corresponding to the considered .pdb file. Precisely, the function scrolls the list of couples considered with a *for* cycle. It uses each couple as the key to read the dictionary produced with the function *Statisticizza()*. It copies the 6-dimensioned numpy array corresponding to the considered dictionary key, in the column index of the selected line. The column index are 6 range indexes, starting from 0 and increasing by 6 for each cycle of the *for* cycle. The array corresponding to the first couple of the couple list goes in the matrix column index 0-5, the array corresponding to the second one in the matrix column index 6-11, …
The function *ModifMatrice()* has no output. It just modifies the content of a matrix given as input, using the information coming from the function *Statisticizza().* The function summarizes the information included in a whole .pdb file in only one matrix line, composed by a number of columns equal to the considered features.

Using the functions introduced above and some other input information, it is possible to create the first version of the training matrix. The necessary input information are the list of considered ligand-protein couples, the considered distance ranges, the starting matrix to be modified.
The list of couples is created starting from the lists of ligand and protein taken into account. The list of atoms belonging to ligand and protein considered in this study, as anticipated, are, respectively:
['H', 'C', 'N', 'O', 'F', 'P', 'S', 'Cl', 'Br', 'I'];
['H', 'C', 'N', 'O', 'P', 'S'].
The atoms belonging to the ligand and to the protein list are all the possible atoms present in ligand and proteins considered in this study . The list of couples is created using a double *for* cycle. The first cycle scrolls the list of considered ligand atoms and the second associates all the atoms of the protein to each ligand atom.
The distance range between the 2 atoms of the couple considered, as anticipated, is a distance of 12 Å, divided in 6 intervals of 2 Å each. Of course it is possible to use

different interval amplitude and total distance. Here we are describing the best descriptive model.

In order to determine the exact dimension of the starting matrix, an initial index is fixed (*Nist*) equal to zero. Then, all the folders containing the .pdb files are opened using a *for* cycle and the command *os.listdir()* creates a list of the files inserted in each folder. The matrix dimension in terms of lines is determined by the sum of the length of each list of files. The initial training matrix composed by the number of lines just found and 360columns, is created.

The process of scrolling the folders containing the .pdb files with a *for* cycle and of creating a list of the files contained in each folder is repeated. Another *for* cycle is used to select one by one the files of the list in order to select the .pdb file to be analyzed and its index. The index is used to select which line of the training matrix has to be modified.

Once the file is selected the previously described functions are applied, according to this sequence, to all the .pdb files contained in the folders: *ModifMatrice*(*Statisticizza*(*DistanzaDaTuttiHETATM*(*openf*()))).

The first version of the training matrix is created.

TARGET VECTOR

The target vector contains the affinity values or the docking scores (if present) of each ligand-protein complex corresponding to the .pdb files used to create the training matrix. The target values are collected in Excel files, as anticipated. The Excel file corresponding to the complexes deriving from a particular protein is contained in folders nominate as the folder containing the corresponding .pdb files. The sheet of the Excel file is named “Foglio1”. The structure of the Excel file is constant. The 1st column contains the .pdb file name, the 2nd column contains the ligand-protein complex code, the 3rd column contains the ligand-protein affinity, the 4th column (if present) contains the docking score in negative value. In order to create the affinity vector, a list of the Excel files is produced with the command *os.listdir().* With a double *for* cycle the folders containing the .pdb files are scrolled, and for each one, the corresponding Excel folder, according to the folder name, is selected. The Excel file is opened and the sheet “Foglio1” is selected. Using another double *for* cycle the .pdb files and the rows of the Excel file are scrolled. The *for* cycle scrolling the Excel files is based on the row index number. The list of row index number is created using the range command and starting from index 2 until index equal to the number of .pdb files contained in the originally selected folder plus one. It starts from index 2 because the first row of the Excel

contains the content title for each column. It is possible that the row index are more than the .pdb files because the Excel file contains more data than expected. In this case an error will occur running the routine and a manual check of the dimension of the Excel file is requested. When the file name reported in the first column of the Excel coincides with the .pdb file name selected, the corresponding target value is added to the queue of the target vector (changing the sign for docking scores). The last *for* cycle, the one scrolling the rows of the Excel file, is stopped. The *for* cycle to read the .pdb files is used in the same way as done in reading the files in the matrix creation process. This guarantees that the corresponding rows of the training matrix and of the affinity vector are related to the same .pdb file.

As anticipated, the case when the Excel file does not include the same ligand-protein complexes present in the folder of the .pdb files can happen. The mismatch can be caused by oversight in compiling the Excel file. For this reason a function is created that checks the correspondence of the analyzed .pdb files with the ones that have the target value in the Excel file.

Function "*trovacopia()*".

Input: the address of the folder containing the analyzed .pdb files and a list in which writing the mismatch files.

Output: the list of mismatching file.

The function *trovacopia()* uses the double *for* cycle to scroll first the list of .pdb file obtained using the command *os.listdir()* applied to the address given as input, and then the rows of the Excel file in the column containing the file name. *Trovacopia()* introduces a list nominated "vettoreConta" which will be used as counter. When the .pdb file selected by the first *for* cycle and the .pdb file name corresponding to the row selected by the second *for* cycle coincide, the file name is added to the counter list. The second *for* cycle is broken and an *if* logic verifies if the counter list is empty or not. When the list is empty it means that the selected .pdb file with the first *for* cycle has no corresponding target value in the Excel. The function *trovacopia()* adds the name of this kind of file to the output list. The rows of the training matrix corresponding to the files contained in the output list has to be deleted from the training matrix

In order to apply the function *trovacopia(),* the previously described procedure to select the folder containing a certain type of .pdb files and the corresponding Excel file is applied. The address for the folder selected is created and the Excel file is opened in

the sheet “Foglio1”. *trovacopia()* is applied to the selected address and to an empty list. If the output of the function remains an empty list, no action is taken. This means that all the .pdb files in the considered folder have a corresponding target value in the Excel. In the case the list contains some .pdb files a double *for* cycle is used. The first scrolls the list and, for each element of the list, the second scrolls a range of index equal to the number of files in the folder initially considered. The indexes are used to consider, one by one, all the .pdb files belonging to the considered folder. They represents the position of the files in the list. Using an *if* logic, when the file name considered, coming from the output list of *trovacopia(),* coincides with the name of a file of the list, the corresponding index is appended to a list called “posizioneErr”. At the end of the cycle analyzing a single folder, the list “posizioneErr” contains the indexes of the .pdb files with no corresponding target values. The list “posizioneErr” is used to delete the corresponding lines from the training matrix. The same procedure is repeated for each folder containing .pdb files, as in the training matrix creation. Of course, before deleting the line from the training matrix, the indexes obtained from the operation just described in a single folder, are summed to the number of the files in the folders previously analyzed minus the number of files already deleted. Once the checking process of the target vector is completed, this latter is saved as a numpy array.

NORMALIZATION PROCESS

The training process uses the matrix created starting from the .pdb files. As already explained, it produces better results if the elements of the matrix are normalized,. The normalization process of the training matrix is done dividing each of the originally 360 features by the maximum value present in the matrix. A numpy array called “Max” is created using the function *np.amax().* The function is applied to the training matrix with respect to the columns (axes = 0). The output of the function is a numpy array containing the maximum value relative to the matrix for each column, so it is a vector 1x360. The function *np.amax()* is applied another time and in the same way to the previously resulting vector 1x360. The result is called “MaxMax” and it is the maximum value present in the matrix. Each element of the matrix is substituted by the same element divided by the “MaxMax” value. The results is a normalized training matrix, in which all the components are included in the interval 0:1.

The target vector is normalized using the functions *numpy.std()* and *numpy.mean()* for calculating, respectively, the standard deviation and the mean value of the considered

data set. Once the standard deviation and the mean value are evaluated, each target value is subtracted by the mean value and then divided by the standard deviation, as described by the formula reported in paragraph 4.1.1.
The normalized training matrix and the affinity vector are saved as numpy arrays.
The described process illustrates the creation of the training matrix and of the target vector which will be used in the neural network training process.

4.2 MLP scoring function neural network and training protocol choice

In the following paragraph the used regression model is presented. The choice of using a multilayer perceptron, is motivated by the ability to provide approximate solutions for extremely complex problems, as predicting binding affinities is.

A multilayer perceptron is an artificial neural network model that maps sets of incoming data into an appropriate set of outgoing data. It is made up of multiple layers of nodes in a directed graph, with each layer completely connected to the next. It consists of at least three layers of nodes: an input layer, a hidden layer, and an output layer . Except for incoming nodes, each node is a neuron (processing element) with a linear activation function (120). The multilayer perceptron is a modification of the standard linear perceiver, described in the first chapter, and can distinguish data that are not linearly separable (121).

A study is conducted to determine the structure of the network which can guarantee the best results.

The study is conducted using experimental data, which is considered the principal database. The structure and training protocol which emerge to be the best are used also with synthetic data. In fact, it is necessary considering that this research was originally envisioned for using a very refined database composed of experimental data. The synthetic data are originally intended as additional data and the scoring function deriving from this pool is initially taken into account as a term of comparison.

The first step to determine the best neural network structure and training protocol is to verify what the best structure is in terms of number of neurons for each layer using a standard training procedure. It means that a basic subdivision of the database into training and test set according to percentage 87.5-12.5 is used. A standard number of 1500 training iterations are performed and only the best R_p measured on the test set is recorded. The training process uses a batch size of 50 and 200 for experimental and synthetic data, respectively. The regularization is constant in each layer and equal to 10^{-6}. The optimization algorithm is ADAM (122).

In order to determine the best structure, considering the total amount of experimental complexes is 2400, layers composed of different number of neurons, even if limited, are tested. In particular, networks with numbers of neurons equal to 10, 20, 30, or 40 are tested. Because of the restricted number of data, a network with limited dimension is expected to be the best. For this reason, networks composed of 2 layers are considered. From this preliminary study, a layer dimension of 20 neurons demostrates to guarantee best results. In fact a network with a layer of ten neurons does not learn accurately the information of the training matrix. In the opposite case, a layer of 30 neurons is not able to generalize the information acquired in the training (reference to Figure 4.3).

Different structures and training protocols are tested and the corresponding results, in terms of Pearson's correlation coefficient, are reported in Figure 4.3:

A. standard network 1x20.

B. Standard network 2x20.

C. Standard network 3x20.

D. Network 2x20 with transfer learning using synthetic data and ligand-protein affinity as target value.

E. Network 2x20 with transfer learning using synthetic data and docking score as target value.

F. Network 4x40 on synthetic data using ligand-protein affinity as target value (phase I) + 1x 20 (phase II).

G. Network 2x20 with transfer learning using synthetic data and ligand-protein affinity as target value, adding a further feature only in the second layer, i.e., the docking score.

The structures and training procedures listed above consider the same attributes specified at the beginning of this paragraph, if not differently specified.

In case A, B and C the standard network is a multilayer perceptron, as previously introduced, composed of an input layer and an exit layer, plus some hidden layers. In the considered networks the number of the hidden layers present are equal to the number reported in the list. The standard network undergoes a single training using the experimental data. The difference among the standard networks ((cases A, B, C) is the structure.

In case D and E the transfer learning technique is applied. The aim is to strengthen the scoring power of the machine learning function even if the training set is small . As anticipated in the first chapter, it consists in performing a pre-training of the network using a database with similar characteristics to the original one. The pre-training has the purpose of modulating the neuron coefficients, to make the final train, with the real data, more effective. In fact, in this case the training of a neural network with a small database, like the experimental database is, risks to produce a function which can predict extremely well the training data. However, the function might not have the capability of extracting general information from the database and making good predictions for new data. Clearly the transfer learning is more effective the more similar the data of the pre-train database are with respect to the real data and if the database utilized in the pre-train is sufficiently larger than the real database. In fact, the synthetic data are used for this purpose. They are more than one order of magnitude more numerous and they are the same type of data with respect to the experimental data. In the case D, the multilayer perceptron is previously trained on synthetic data considering the ligand-protein affinity as target value. The pre-train lasts until the mean squared error calculated on the validation pool stop decreasing for ten iterations consecutively. The coefficients of the network obtained up to that moment are used as initial parameters to start the final training on experimental data.

In case E, the same training procedure of case D is applied. The only difference is that the target value used in the pre-train, for synthetic data, is the docking score.

The network described at point F is built in two phases. In the first phase it consists of a multilayer perceptron composed by 4 layers of 40 neurons. Here the network is trained on synthetic data considering the affinity between proteins and ligands as target value. In the first phase, the training ends when the loss function measured on the validation set does not decrease for 10 iterations consecutively. In the second and last phase a further layer composed of 20 neurons is added to the network. Here the effective training on experimental data is performed keeping locked the coefficient obtained in the pre-train for the first 4 hidden layers.

In case G another technique, again based on transfer learning, to possibly increase the performance of the machine learning scoring function is applied. It consists in using the standard network 2x20 adding a further feature to the training matrix only for the last layer in the pre-train. The additional feature is the docking score. As in previous cases, the pre-training phase lasts until the cost function does not decrease for 10 iterations consecutively. Then, the final training, on experimental data, is performed.

The plot in Figure 4.3 represents the difference performances of the MLP scoring function, in terms of R_p, for the network structures and training protocols just described. In the end, the strategies used to increase the scoring power of the machine learning scoring function do not produce promising results. Because of this, the simplest solution is chosen, namely a standard multilayer perceptron of size 2x20 for the experimental data. While a deeper network of size 4x40 is used for the synthetic data.

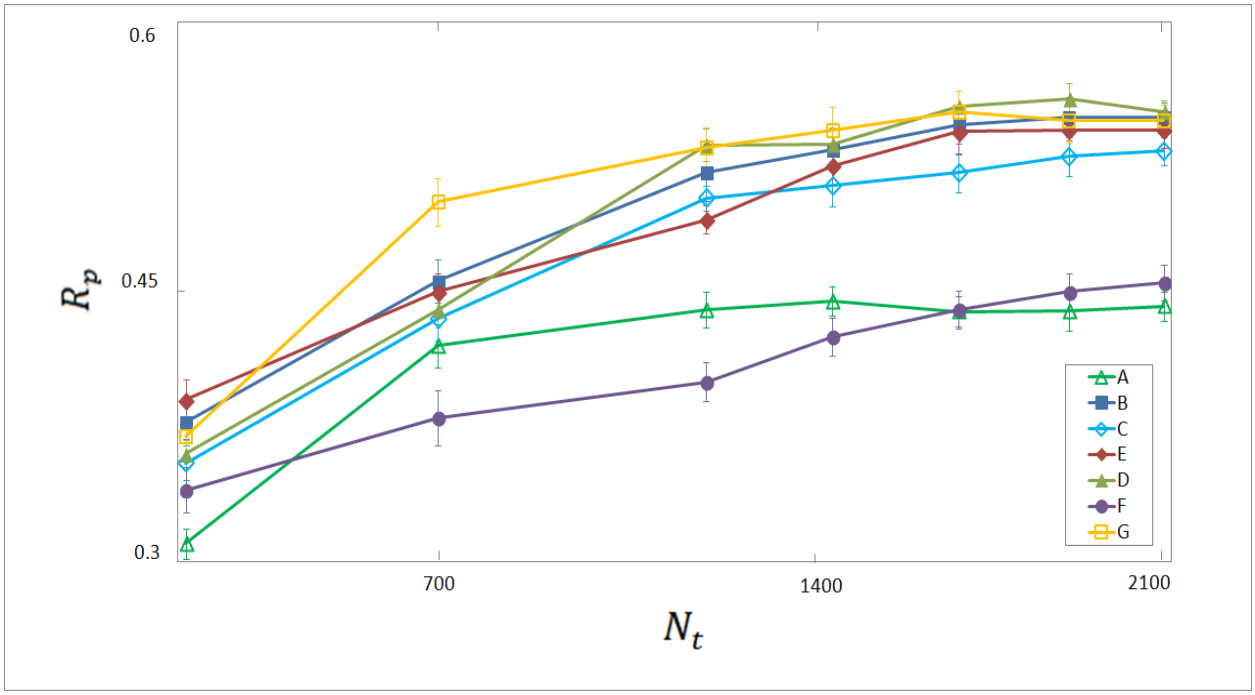

Figure 4.3 Neural network structures and training protocols tested using the different protocols described on Paragraph 4.2

4.2.1 MLP scoring function: code analysis

In the following paragraph the process of creating the MLP scoring function is presented through the code analysis. This latter is analyzed for each network type we consider, merging the similar parts in a single analysis. All the procedures are implemented with Python.

Training matrix and target vectors loading

The creation of the MLP scoring function starts with the training process. In order to do it the training matrix and the target vector, already created, as previously described, are loaded with the command *np.load()*. In the case the transfer learning technique is applied, both the synthetic and experimental matrix, with relative target values, are loaded.

Superfluous features elimination

The descriptive model used in this study, as anticipated, considers a total area of radius 8 Å in which the atomic couples are counted. The considered intervals measures 2 Å. This training matrix is originally created considering a total area of radius 12 Å. Because of this, after loading the training matrix, the code expects the elimination of the features in the ranges 8- 10 Å and 10 - 12 Å. For doing this, the indexes of columns containing data of the intervals 8-10 Å and 10-12 Å are included in a list. The indexes are created using a *for* cycle. Then the columns are deleted from the training matrix using the command *np.delete()*. The function is applied to the original matrix. The indexes of the columns to be removed are in the just created list. The axis chosen for applied this command is axis 0 (vertical).

Training matrix normalization

After this preliminary operation, the maximum of all the features considered is calculated. If the transfer learning technique is applied, the maximum of the features is calculated both for the synthetic and the experimental matrix. The function used for finding the maximum of the features is *np.amax()*. This function is applied along the axis = 0, namely the vertical axis. The function *np.amax()* is applied twice. The first time it is applied to the training matrix and the maximum for each feature is found. Then the function is applied to the deriving one dimensional vertical vector, in order to find the overall maximum.

The absolute maximum is used to normalize the training matrices. Each element of the matrix is divided by the just obtained relative maximum. In this way the training matrix is composed of 240 features, the relative value is in the interval [0, 1].

PRE-TRAINING PHASE

Training matrix and target vector permutation

In the cases the transfer learning technique is applied, the single cycle of training is started using a simple *for* cycle. In fact we will see in the next paragraph that in this research each result is calculated as an average value on 10 different trainings. The code analysis conducted from this point is inside this cycle.

After the beginning of the training cycle, a permutation of the pre-training matrix and of the relative target vector is performed. The permutation is performed in the following

way. An index list from 0 to the total number of rows present in the pre-training matrix is created. The index list is permuted using the command *random.sample()* applied to the index list. The command is applied in a way that each index can appear only one time in the list. The resulting permuted index list is used to reorder the matrix and the target vector in the same way. The aim is to maintain the correspondence between the matrix row and the relative target value.

Outliers elimination

The next action is the elimination of the outliers, in terms of ligand-protein affinity. In fact, the synthetic data are numerous in quantity and checking each complex is not efficient. However some errors can appear in the database creation, as previously presented. In particular some molecules can have a negative or out of range ligand-protein affinity or docking score because of some errors in the transcription of the values. For this reason the target values out from the interval 2-22 are considered outliers. This range of intervals is considered because 99% of the complexes are within the interval. The index of these complexes is identified and the corresponding row of the pre-training matrix and of the target vector is deleted.

Target vector normalization

The code analysis continues with the normalization of the target values for the pre-training data. The target values distribution must have the mean equal to 0 and the standard deviation equal to 1, as previously described. The reason for this choice is to have homogeneous distribution if different databases are used in the training of the same network, like in the case when the transfer learning technique is applied. For this reason the mean and the standard deviation of the target values for the pre-training data is calculated. Then each target value is subtracted by the mean of the target values just obtained. Eventually the result is divided by the standard deviation of the just obtained target values. In this way the new target values distribution has a mean equal to 0 and a standard deviation equal to 1.

Training, validation and testing pool

At this point, the data inside the database are ready to be used in the pre-training phase. The data needs to be subdivided into different ensembles to correctly complete the network training. In particular, a training, a validation, and a testing pool need to be

created. The training pool contains the data used to train the network. For this reason this is the larger ensemble among the 3. The validation pool contains a little amount of data used to check the training procedure. In particular, these data are used apply the early-stop criterion. The early-stop criterion indicates the condition that, if verified, implies the strop of the training procedure. In this case the pre-training phase is stopped by the *EarlyStopping()* function. The early-stop option is a callbacks function. It is initialized just after the creation of the pool by the *callbacks()* function. In this function, the parameter to be monitored for the early stopping and the patience rate have to be specified. The parameter indicates what function is monitored. In particular, the decreasing of this parameter is monitored. The patience rate indicates the number of continuous iterations for which the parameter selected can increase before stopping the training. In this study the parameter used for applying the early-stop function of the pre-training phase is the mean squared error measured on the validation data. The patience rate is 10.

callback = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10).

The testing pool contains the data used to test the obtained function. These data are the ones used to measure the final performances of the created function. In the analyzed case, namely the pre-training phase, the testing data are just used to evaluate the pre-training effectiveness. The size of each pool can depend on the particular type of training. In the pre-training phase the synthetic data are used. The testing data are 1500 complexes, the validation data are 1000 complexes and the remaining complexes are used as training data.

Neural network structure

The following phase of the neural network training is the creation of the neural network structure. The structure is different according to which type of network is analyzed. Considering that, in this subsection, we are presenting the pre-training phase, in this paragraph only the structures of the network that use the transfer learning technique are analyzed. In particular we are referring to network types D, E, F, and G.

First of all the model of the network (sequential, API,..) is specified, together with the proper name we assigned to the network. A *Sequential* model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. Otherwise, the Keras functional *API* is a way to create models that are more flexible than the *Sequential* model. The functional *API* can handle models with non-linear topology, shared layers, and even multiple inputs or outputs. In cases D, E, and F the

sequential model is used. In these cases the network is created with the *add()* method. Therefore it is possible to add as many layers as needed. In case D and E, 2 layers are added. In case F, 4 layers are added. Each layer is accessible via the layers attribute. The layers used in the network are Dense*.* Dense layers implement the operation:

$$output = \text{activation}(\text{dot}(input, kernel) + bias).$$

where $activation$ is the element-wise activation function passed as the activation argument; $kernel$ is a weights matrix created by the layer. $bias$ is a bias vector created by the layer; *dot()* indicates the dot product. The activation function used is the hyperbolic tangent (*tanh*). Each layer uses an exit regularization (*kernel_regularizer=regularizers.l2*) equal to 0.000001. Also the regularization is an attribute of the layer. The number of neurons present in each layer is specified in the attribute too. The numbers used in the different cases presented during the pre-training are:

D: 2 layers of 20 neurons each.

E: 2 layers of 20 neurons each.

F: 4 layers of 40 neurons each.

The first dense layer has the input dimension (*input_dim*) of the training matrix as a further attribute. The matrix dimension consists in the number of columns, namely the total number of features which describe an instance. For this reason, as already specified, the input dimension is 240. The exit neuron has to be specified too. It is treated like the previous layers, just avoiding the specification of the activation function.

In the case G, a model with non-linear topology with multiple inputs is created. The first action is the specification of the various inputs. In case G the specified inputs are called *feature_input* and *dk_input*. The first input represents the standard 240 features chosen to describe the molecules. The second one is the docking score which is used as an additional feature, directly connected to the last layer of the pre-training. In each of these inputs, the dimension of the matrix in terms of columns must be specified in the attribute *shape.* After this phase the neural network structure is built in terms of layers and the relative inputs are specified. Here 2 dense layers of 20 neurons are used. The first layer is inserted with the command *layers.Dense()* and called *feature_features.* The number of neurons and the activation function are specified like its attributes in this way: *feature_features = layers.Dense(20, activation="tanh")*. The number of neurons is 20

and the activation function is *tanh.* The input of the layer is specified successively to the layer attributes inside round brackets. In the case of the first layer, the input is *feature_input*. Then the output of the first layer is concatenated to the *dk_input* with the command *layers.concatenate* and is called *x*: *x = layers.concatenate([feature_features, dk_input])*. The second layer has the same attributes of the first and its input is the vector x. Eventually the exit layer is created with the command *layers.Dense* too and called *priority_pred*. The number of neurons is 1 and the activation function is *tanh*. The input of this last layer is the output of the previous layer, called *x* again. Here is the structure.

```
#Input
feature_input = keras.Input(shape=(240,), name="features")
dk_input = keras.Input(shape=(1,), name="dock")

#Layer 1
feature_features = layers.Dense(20, activation="tanh")(feature_input)

#Additional feature
x = layers.concatenate([feature_features, dk_input])

#Layers 2
x = layers.Dense(20, activation="tanh")(x)
priority_pred = layers.Dense(1, activation="tanh", name="priority")(x).
```

The model just built is created with command *keras.Model()*. The network is called NNbypass. Inside this command the inputs and the outputs of the model must be specified as attributes. In our case:
NNbypass = keras.Model(inputs=[feature_input, dk_input], outputs=priority_pred).

Network pre-training

Whatever is the model created, it is configured for the pre-training with the command *compile()*. This command expects some attributes which enable the configuration of

some training settings. In the case of MLP scoring function the attributes specified are the following: *(loss='mean_squared_error', optimizer='adam')*

loss indicated the loss function. It is the value minimized by the model. In our model we choose the mean squared error. It is calculated between the real and the predicted value.

optimizer indicates the optimization algorithm. (It permits us to find the weight values for which the cost function is minimized). Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.

The pre-training starts a built-in training loop (the *fit()* method). The function *fit()* receives the following attributes.

- The input and the target data for the training.
  They are the training pool previously created.

- The number of epochs.
  The quantity used in this study is 1500 (*epochs = 1500*). The total number of 1500 epochs is reached only if the early-stop criterion is never realized before.

- The batch size.
  It is the number of samples per gradient update. In the case synthetic data are used, like in the pre-training phase, analyzed here, the batch size is 200 (*batch_size = 200*).

- The callbacks.
  It is a list of callbacks to apply during training. In this case the previously presented callbacks, called *callback,* are applied. They are just used for the early stopping option application (*callbacks=[callback]*).

- The validation data.
  The validation set in terms of input and target data are indicated. This set is used to apply the early-stop criterion. In this case the validation pool previously created is furnished (*validation_data=(x_val, y_val)*).

End of pre-training phase

Once the pre-training phase ends, either because the early-stop criterion is reached or because the total number of epochs are performed, the final training can start. The final

network in the cases presented in this study, is built with the *Sequential* model. A name is chosen for the network. The structure is created, like described before, with the *add()* method. In case D, E and G, 2 dense layers of 20 neurons are added. In case F, 4 dense layers of 40 neurons plus 1 of 10 neurons are added. The attributes of each layer are the same used in the pre-training phase. The activation function used is the hyperbolic tangent and the exit regularization is 0.000001. The first layer has the input dimension as a further attribute and the exit layer does not have the activation function attribute. Only a further attribute is used in case F. In fact, in this case the weights obtained by the pre-training in the first 4 layers are copied and kept locked in the final training. For this reason, in the final network, the first 4 layers have the additional attribute of being not trainable (*trainable=False*). The weights of the first 4 layers are copied from the ones obtained in the pre-train phase. Then they are set up in the first 4 layers of the final network. The commands used to do it are *set_weights()* and *get_weights().* They are applied to respective layers using the index notation.

FINAL TRAINING

Training and testing pool

In the final training the database is divided into training and testing sets. The validation set is usually not present because the training completes all the epochs specified in the *fit()* function. We are not using *EarlyStopping()* in the final training In each epoch the performance parameters are recorded. In this way, the trend of the performance parameters is recorded along the entire training process. In the final training the experimental data are used. The testing data are 300. The remaining data are used as the training set.

Neural network structure

As already specified, the network used in the final training is built with the *Sequential* model in the way previously described also for case A, B and C. In case A, B and C, 1, 2 and 3 dense layers of 20 neurons are added respectively. In the remaining cases the network structure used is previously specified.

Network final training

The final training starts with the use of the function *compile()* applied to the final network. Like in the pre-training phase, the loss function and the optimizer is specified.

They are the same used in the pre-training phase. Then the function *fit()* is applied to the final network, as in the pre-training phase. In this case, the input and the target data, for the training, are experimental data. The quantity of data used is previously specified. The batch size in this case is 50. The rest of the attributes are identical to the ones used in the pre-training corresponding phase. The entire number of epochs specified in the attributes of the *fit()* function is completed in the final training and the performance parameters for each epoch are recorded, as already said. In this way the maximum of the performance parameters used are certainly recorded. In fact, the best performing MLP scoring function, on test data, is always obtained before the 1500th epoch. On the other hand, if the *early-stop* function is used, the training process is more efficient, but sometimes it can stop the training before the best performance of the function is reached.

Target data prediction

The prediction of the target data is done using the function *predict()* applied to the testing input data in the final network. Usually, in addition to the testing data, the prediction is applied also to the training input data to verify the effectiveness of the training process.

Network performance parameters

Once the prediction vector is created, the performance parameters for evaluating the scoring function created, are calculated. In our study, as already specified, the performance parameters taken into account are: R_p on the train set, R_p on the test set, R^2 on the train set, R^2 on the test set, Mean Absolute Error (MAE) on the test set, and mean squared error (MSE) on the test set. The mathematical meaning and properties of these parameters are already treated. In the code, the parameters are evaluated using the following expression:

R^2 in the train set (*Rsqtrain*):

msqetrain = mean_squared_error(enmytrain, y_pred_train)

vartrain = np.var(enmytrain,ddof=1)

Rsqtrain = 1-msqetrain/vartrain

enmytrain is the real target vector for the train set.

y_pred_train is the predicted target vector for the train set.

The function *mean_squared_error()* belongs to the *sklearn.metrics*. It calculates the mean squared error between 2 vectors using the mathematical expression already reported in the text.

The function *np.var()* computes the variance along the vector *enmytrain. ddof* indicates the "Delta Degrees of Freedom". The divisor in the mean calculation is the total number of elements of the array minus the *ddof*. In standard statistical practice, *ddof*=1 provides an unbiased estimator of the variance of a hypothetical infinite population.

R^2 in the test set (*Rsqtest*):

msqetest = mean_squared_error(enmytest, y_pred_test)

vartest = np.var(enmytestEXP,ddof=1)

Rsqtest = 1-msqetest/vartest

enmytest is the real target vector for the test set.

y_pred_test is the predicted target vector for the test set.

The procedure is analog to the calculation of the R^2 in the train set.

Pearson Correlation Coefficient in the training set (R_p):

R_p *= pearsonr(enmytrain, y_pred_train)*

The function *pearsonr()* is a function of the module *scipy.stats* which calculates the Pearson Correlation Coefficient between 2 vectors automatically according to the mathematical expression already reported in the text. This module contains a large number of probability distributions, summary and frequency statistics, correlation functions and statistical tests.

Pearson Correlation Coefficient in the test set (R_p):

R_p *= pearsonr(enmytest, y_pred_test).*

Mean absolute error in the test set (MAE):

maetest = mean_absolute_error(enmytest, y_pred_test)

The function *mean_absolute_error()* belongs to the *sklearn.metrics*. It calculates the mean absolute error between 2 vectors using the mathematical expression already reported in the text.

Mean squared error in the test set (MSE):
msqetest = mean_squared_error(enmytest, y_pred_test).

All the performance parameters calculated after each epoch are recorded in a list, till the training is ended at the 1500th epoch. The performance parameters are recorded as a row-element of a list. The row-elements are recorded using the command *append()*. Because of this they are recorded sequentially, from the first to the last element. Each row-element contains six elements which correspond to the 6 performance parameters just described. At the end of the training process the list is transformed into a *numpy* array (*np.array()*) in order to better manage the file and its values. The training procedure ends with the save of the array obtained in which the performance of the function created is reported. Then the maximum or the minimum, as needed, can be calculated for each parameter, using the function *np.amax()* or *np.amin().*

4.3 Test set choice

To guarantee reliable results in the MLP scoring function performance test, it is important to choose a method that guarantees equal conditions for each test. In this paragraph we present the way used in this study to measure the performance.

The performance of the MLP scoring function is measured on the test set. This latter is a subset of the database not used for the training and validation operations. As already presented, the database, in particular the synthetic one, is created considering sequentially complexes coming protein by protein. Choosing an interval of the database, especially for the horizontal test, means performing a test that is not reliable. In fact, the chosen subset can contain complexes derived by a single protein or few proteins, possibly never seen in the training. In this case a horizontal test can become a vertical test. Another possible case is that the MLP scoring function can predict very well the complexes deriving by some specific proteins and in the test set are selected complexes deriving by these proteins. The presented cases produce unreliable performance results. In order to avoid these situations the permutations of the training matrix and target vectors is performed, as previously described, before subdividing the

database into the subsets. The use of the permutation for the database cannot prevent all performance measurement mistakes. In fact, if a single test is used to measure the performance of MLP scoring function, even if the database is permuted, it can encounter particular favorable or unfavorable conditions. These conditions are simply due to the subdivision encountered in the subsets creation. In order to avoid this situation, in this study, for each performance parameter, we consider the mean value on 10 different tests. A different test corresponds to a different permutations of the database. In addition to the mean performance value, also the standard error is calculated for each parameter according to the following expression:

$$STD.ERROR = \frac{\sigma(R_p)}{\sqrt{10-1}},$$

where $\sigma(R_p)$ is the standard deviation of the R_p calculated in the 10 different tests.

5 MLP scoring function performance

In this chapter we describe the performance of the scoring function created for this study, namely the MLP scoring function. The value of R_p considered here is computed as the mean value on 10 different attempts. In fact, as discussed before, in each attempt a random permutation of the matrix and the corresponding target vector is applied before the training.

5.1 Horizontal test

In Figure 5.1 the performance of the MLP scoring function for a horizontal test is reported. We consider 3 different scoring functions, according to the database and the target values used to train the network. The test set is composed by 300 and 2000 complexes, respectively, for experimental and synthetic data. These numbers represent the 12.5 % and the 7% of the corresponding databases. As discussed above, in the present study, two different scoring functions are analyzed, one trained on synthetic and one on experimental data. In particular, for synthetic data we consider two scoring functions, one for predicting the binding affinity and one for predicting the docking score. The function used to forecast the docking score is used to check the goodness of the model proposed by this study in terms of complex descriptive model and training procedure. In this last case, a maximum of R_p 0.85 is reached. According to the correlation ranges described in Chapter 1, it means that a strong correlation occurs between real and predicted scores. This is a proof that the model used in this study is reliable. The descriptors used are sufficiently complete to guarantee, at least, the performance of a classic scoring function, such as GOLD (59). Generalizing, it seems that the scoring power of commercial scoring functions can be easily emulated by machine learning scoring functions, but this is another subject that will not be treated in this study.

In the case of the MLP scoring function built with experimental data, we obtain R_p = 0.55 in the horizontal test. In this case, only a moderate correlation is obtained between real and predicted ligand-protein affinity. In fact, this type of correlation occurs when the R_p is in the interval between 0.3 and 0.7. Also when the synthetic data are used a

moderate correlation is present, in fact we obtain R_p 0.60. The performance of MLP scoring function is similar to the one of other machine learning scoring function, as presented at the beginning of this study. The MLP scoring function created using synthetic data can reach higher performance than the one with experimental data. Even if the synthetic data are about one order of magnitude more numerous with respect to experimental data, and this is an important factor when you use deep learning, they have less reliable3D structure in the binding site. The descriptive model used in this research, as already presented, is based on the 3D molecular structure. For this reason, the result can be surprising. However, we consider the hypothesis that a machine learning scoring function is able to recognize similar proteins instead of complex binding proprieties, as discussed by Jincai Yang *et al* (118). In this case, it is easier to find similar proteins in a pool where the variance of the proteins is sensibly inferior. This happens in the synthetic database, which has a smaller variance than the experimental database. In fact, the MLP scoring function, in the case of synthetic data, receives a train on 17 proteins complexes. Indeed, each group is composed of a number of complexes almost of the same order of magnitude of the entire experimental database. In the case of experimental data, the proteins taken into account are sensibly more numerous, considering that each protein is present in about ten complexes. For this reason, the difference between the R_p obtained with experimental and synthetic data is not surprising. It confirms the theory that a machine learning scoring function learns to recognize the proteins and not the molecular binding mechanisms.

The gradient of the curves of experimental and synthetic data as a function of the training set size are different, as one can see in Figure 5.1. There, the R_p on the y axis is plotted as a function to the training set size on the horizontal axis. This could be explained with the argumentations presented before. The experimental data have a big variance in its domain. Because of this, when you train the network on a small train set, it is probable that train and test set have a small overlap in terms of protein types. As already discussed, similar complexes present contemporary in train and test set increase the performance of the scoring function in terms of R_p. Increasing the train set dimension, the overlap phenomenon increases and consequently the R_p score improves.

5.1.1 How the percentage of database used in training influenced the horizontal test

In Figure 5.2, we plot R_p reached by the MLP scoring function as a function of the percentage of the training set compared to the whole database. Differently from the previous figure, the behaviors of the functions built with the different types of data are similar in terms of slopes. In addition, the qualitative behavior is reasonably well approximated by a straight line.

As a consequence of all the observations proposed, it is clear that what matters in the R_p increase for a machine learning scoring function, whether built with experimental or synthetic data, is the percentage of the entire database on which the network is trained and not the binding site refinement.

The maximum R_p obtained in a scoring function horizontal test prediction is influenced by the overlap of proteins present in train and test sets. Generalizing, the similarity between complexes in train and test set increases performance measurement in horizontal test. Considering Figure 5.2, one understands that the higher the overlap is, the larger the vertical shift of the straight line is.

This is a confirmation of Jincai Yang *et al* (118) thesis. They sustain that a machine learning scoring functions learn to recognize the different proteins and not the complex bounding features. Moreover, similar databases used in train and test are a doping factor in the R_p measurement.

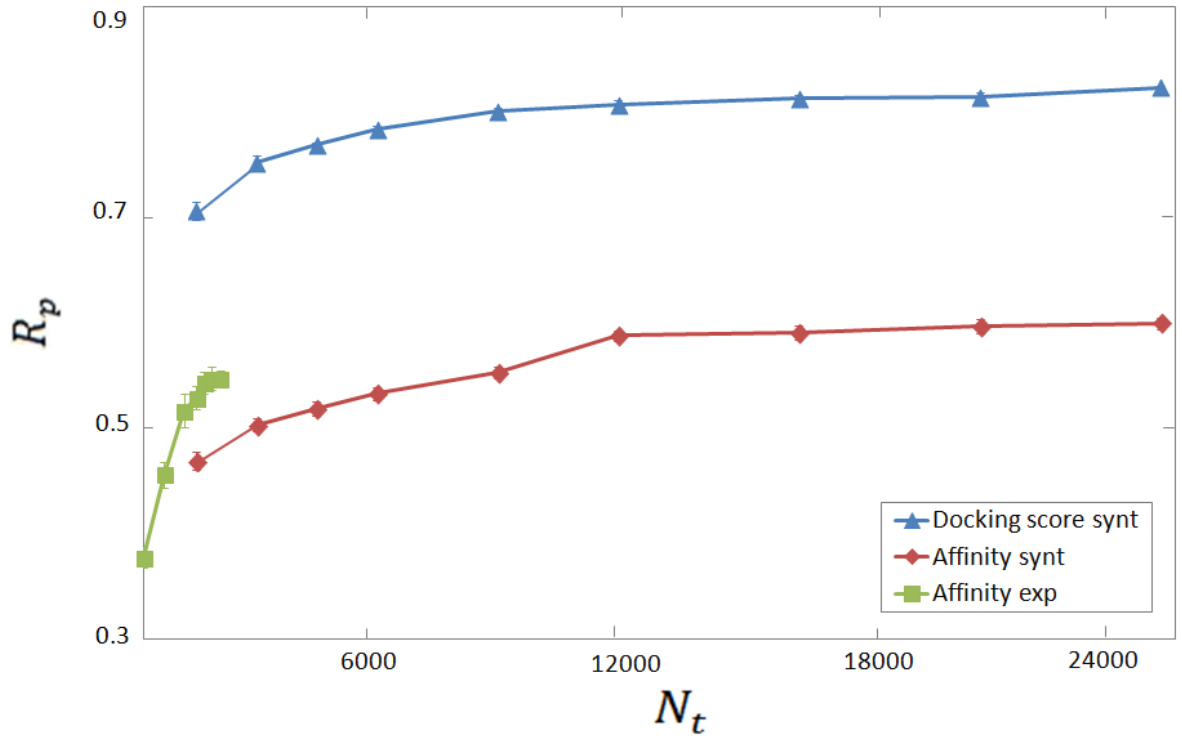

Figure 5.1. MLP scoring function performance for trainset dimension on horizontal test using the three different databases for relative training and test

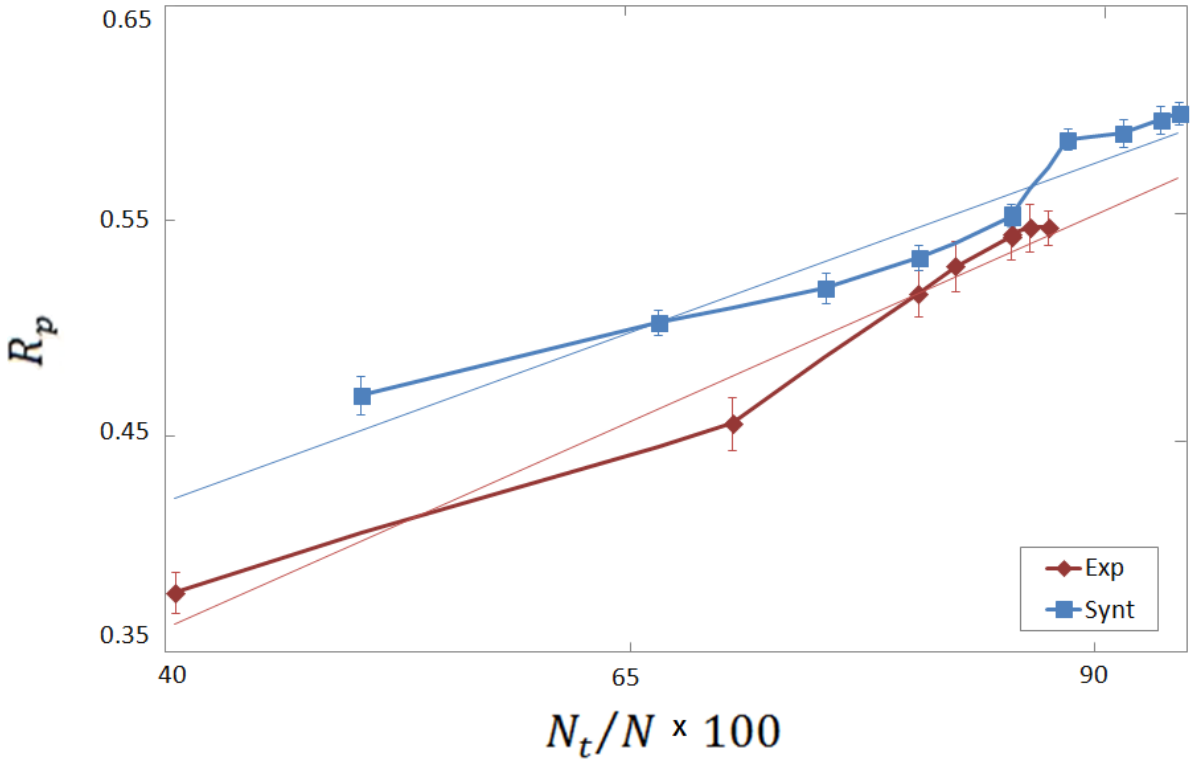

Figure 5.2. MLP scoring function performance as a function of the trainset percentage with respect to the complete database on horizontal test. The database used are the experimental and synthetic one. The target value considered is the ligand-protein affinity.

5.2 Vertical test

In order to verify the real performance of the MLP scoring function, we perform more stringent tests, which can describe a real-case scenario. In particular in chemical and pharmacological laboratories, the typical challenge is to predict the affinity between a protein, which was never addressed before, and some ligands.

With these premises, a horizontal test does not describe the situation well, because the machine learning scoring function is tested on complexes similar to others already seen. In particular, the protein in your test is already experimentally measured in the reactions with many other ligands. A vertical test is the one we are looking for in this scenario. In fact, in a vertical test, the network is trained on complexes deriving by some specific proteins. The consequent test is performed on a pool of data which are not deriving from any protein present in the train set. This is the vertical test. In Figure 5.3 it is possible to observe that a vertical test, using the MLP scoring function trained on synthetic data, produces a performance in terms of R_p sensibly inferior to the one measured in the horizontal test. In addition, a constant value of R_p is measured when the dimension of the training pool is increased. In the case described here, the test set is composed of 2068 complexes made from 4 different proteins. The protein considered in this test are: FAAH, PIM2, ACE, and MCL1. Each protein counts, respectively, 508, 384, 488, and 688 complexes. The complexes of these specific proteins are chosen because they are the less numerous pools among the ones present in our database. The intention is to maximize the variety of proteins, minimizing the dimension of the test set. The remaining database is used for training the scoring function. The training set contain complexes deriving from the 13 proteins not used in the test set. For this test, the data considered are synthetic, because of the possibility of easily dividing the training and test set among different protein pools.

The results obtained by MLP scoring function in a vertical test, even if sensibly lower than the one in the horizontal test, still show a moderate positive correlation. The result is better than the one obtained by Wójcikowski *et al* (98), which is the other team that performs a similar test. They obtain R_p = 0.2.

It is important to notice that the scoring power of the function measured in term of R_p do not increase with the train set size. This confirms the thesis, already presented, that machine learning scoring function have good performance if the affinity prediction is made for complexes similar to the ones used in the training phase. In fact, in a vertical test, by definition, similar complexes deriving from the same proteins are not contemporary present both in train and test set. A machine learning scoring function develops the ability to recognize similar complexes in terms of proteins or ligands. When the affinity prediction is done on complexes totally different from the ones used in the training, the result is degraded.

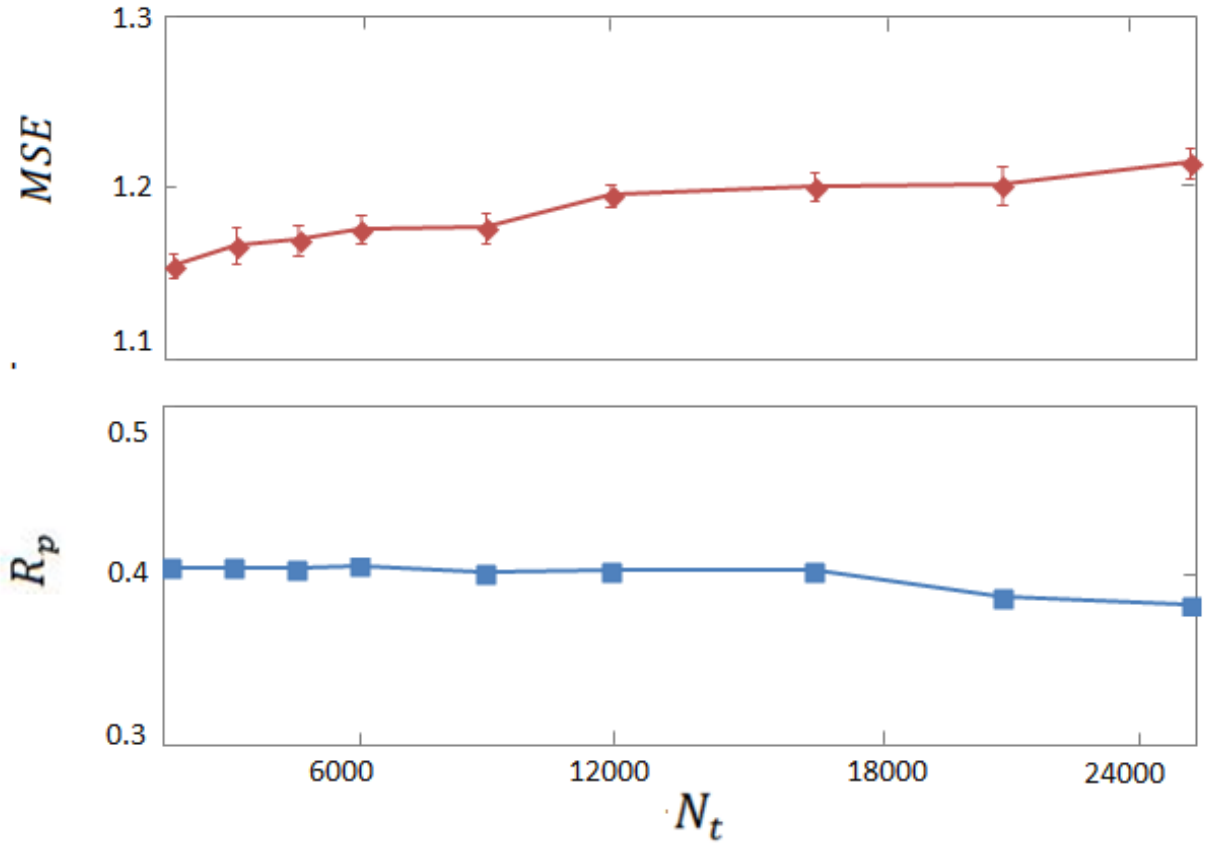

Figure 5.3 MLP scoring function performance for trainset size on vertical test. The database used is the synthetic one. The target value considered is the ligand-protein affinity.

5.3 Per-target vertical test

In Figure 5.4 another type of vertical test is performed, namely, what we refer to as the per-protein vertical test. The MLP scoring function is trained on the entire database

except the complexes deriving by one protein. These are used as test set. Using this type of test a typical scenario of a new medicine discovery, where by the target is previously selected, but never studied before, is described. In this plot, the protein names reported in the abscissa are the protein complexes used as test set. The train set is the remaining database excluding the complexes of the test set. The dimension of the test set and, consequently, of the train set, varies depending on which protein complexes are considered. As in the previous case, the data considered are synthetic.

The performance of the function is similar to the one described in Figure 5.4, providing a confirmation of the argument presented above. In fact, the mean R_p is degraded with respect to the horizontal test. Considering each protein vertical test, an average behavior describes a very moderate correlation between the data predicted and the real one (R_p = 0.35). The exception is made by the complexes of the proteins JAK1 and JAK2. Indeed, their affinities are predicted in a way that guarantees an almost strong correlation with real data. The explanation can be found in the high sensibility to the ligand molecular weight of these two proteins. Most part of the scoring functions are sensible to the molecular weight. It means that a scoring function tends to predict a higher affinity with the increase of the weight of the ligand. In fact, this trend can be really observed. However, it is normally verified only for a restricted range of values and it is not a rigorous law. The MLP scoring function can learn this simple trend and just apply it in the case of proteins JAK1 and JAK2, as shown in Figure 5.4. In fact, in the case of these proteins, the bounding task is large and the just described simple approximation, has a higher range of validity. This is the reason why the sensibility to the molecular weight of JAK1 and JAK2 is higher.

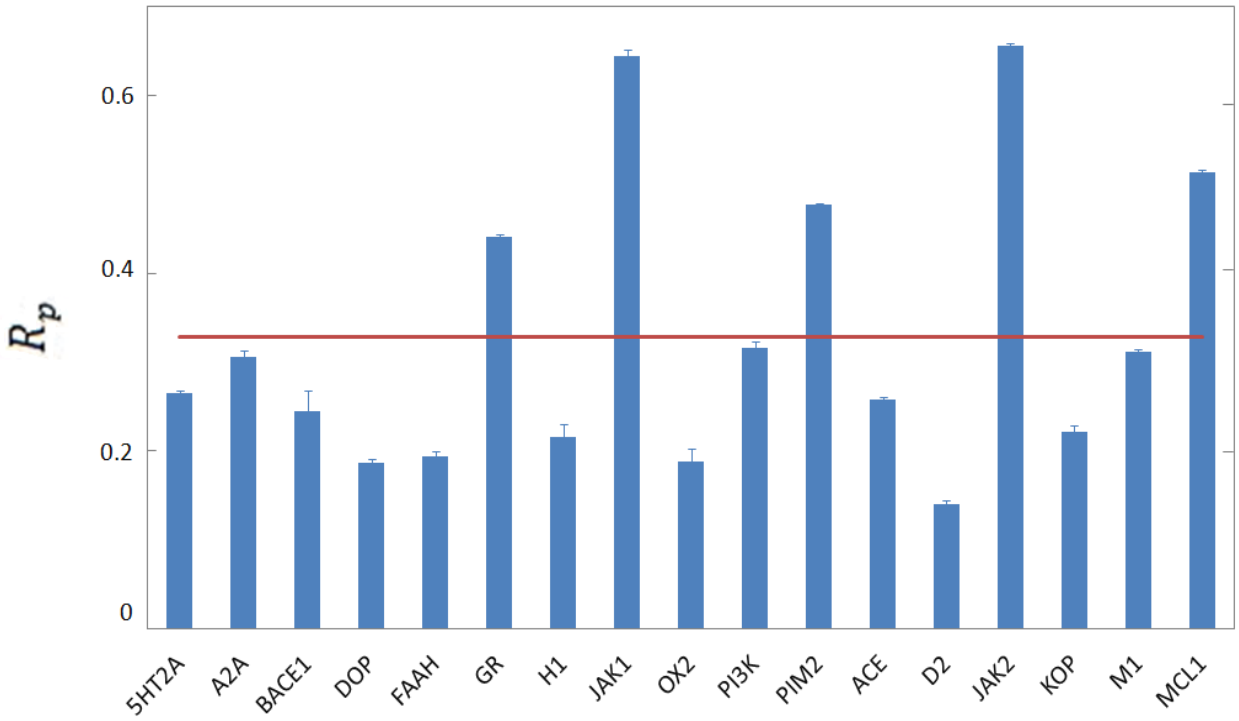

Figure 5.4 MLP scoring function performance on per target vertical test. The database used is the synthetic one. The target value is the ligand-protein affinity.

5.4 Performance comparison

The results obtained from the vertical and per-target vertical test confirm that the use of a machine learning scoring function, which guarantees good performances in horizontal test at the point that the correlation between real and predicted data is moderately strong, seems to be mildly useful in chemical and pharmacological applications. The reason is that chemical and pharmacological typical scoring functions applications contexts are reproduced by the vertical tests. In this scenario, the use of machine learning scoring function shows a degraded performance with respect to the results obtained by the same in a horizontal test. In the case of MPL scoring function the R_p score decreases by at least 0.2. The correlation between real and predicted data, reached by the machine learning scoring function downgrades from moderate strong to moderately weak or weak. The degradation of performances is confirmed by further studies on the subject (118) (98) (103).

6 A possible solution: per-target scoring function

The intention of making a step forward in terms of machine learning scoring function leads us to explore the field of per-target scoring functions.

There are some factors that push the team toward the idea that the creation of individual scoring function for each target can be a suitable solution in the field of scoring function based of machine learning techniques. One factor is the difference measured among the R_p in each protein, in the per-target vertical test. The other factor is the awareness that a single scoring function can perform well or not depending on the target protein considered, as already emerged by the study of classic scoring functions.

Besides, the following reasons encourage us in developing all the potential of synthetic data: the easiness in creating synthetic data; the large quantity of synthetic available data; the performance of the MLP scoring function obtained using synthetic data. A machine learning per-target scoring function is a scoring function created using a neural network for which the training and testing data come from an individual target. This solution can be effective because it avoids the machine learning scoring functions tendency of learning to recognize similar proteins when the training is performed in a database with many different of them. For this reason, the machine learning per-target scoring function strengthens the function capability of learning the propriety of the complex binding mechanism.

The neural network used in this study is a multilayer perceptron, as for the MLP scoring function considered above. This scoring function is called MLP per-target scoring function, where the word "target" is substituted by the target considered. In the cases shown in Figure 6.1, all the per-target scoring functions use a network structure of 2x20 except for the one created for protein D2. In fact, these protein pools have a number of data that vary from 1200, for protein JAK2, to 3000, for protein A2A. The dimension of the datasets is close to the one of the experimental data. Therefore the structure 2x20 is used. The MLP per-D2 scoring function uses a 3x20 structure considering the number of data available for this protein is 6500.

The data used to train the function, as anticipated, are synthetic. The molecular structure is created by the docking software MOE (54) starting from the original 3D structures of the protein and the ligand. The ligand-protein affinity is experimentally

measured. The process of measuring it is simpler and quicker than a crystallographic radiography measurement of the 3D complex structure. In addition, the real ligand-protein affinity is available in shared databases for a very larger number of complexes with respect to the quantity of available experimental crystallographic structures.

The MLP per-target scoring functions reported in Figure 6.1 use a test set of 300 data. The choice of the 6 proteins considered in the ensemble of 17 available is simply due to the fact that they are the more numerous pools deriving from a single protein. The plot of R_p for the MLP per-D2 scoring function is not completely visible in Figure 6.1. The reason is that it has a larger database with respect to the others. The final R_p obtained for a training set of 6200 data is R_p= 0.41. As it is possible to observe in Figure 6.1, the performances, in terms of R_p , of the MLP per-target scoring functions considered, are all better than the MLP scoring function in a vertical test and in the corresponding per-protein vertical tests. This is noticeable by making comparison with the performances shown in Figure 5.3 and 5.4.

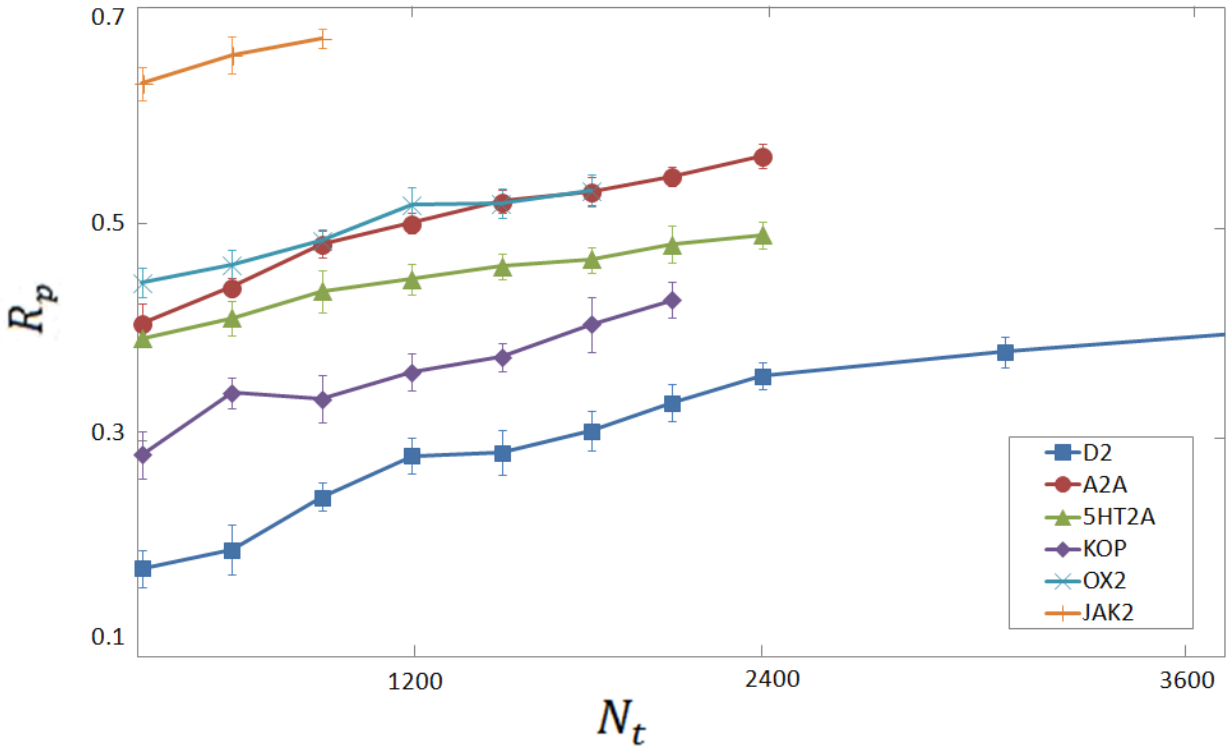

Figure 6.1. MLP per-target scoring function performance as a function of the training set size considering different proteins. The database used is the synthetic one. The target value considered is the ligand-protein affinity.

6.1 Comparison among different types of scoring functions on the same test set

In Figure 6.2, one can see a homogeneous comparison between scoring functions. In fact, here the employed test set, in terms of size and in terms of target used for the complexes, is the same for each group of functions: the MLP per-target scoring function, the MLP scoring function in a per-protein vertical test and a simple molecular weight scoring function. A molecular weight scoring function, as previously described, is a simple linear fit of the ligand-protein affinity as a function of the ligand molecular weight, using a least-squares regression. Two data pools are considered as test set, one composed by protein OX2 complexes and the other by protein JAK2 complexes. In both cases the comparison shows better performance of the per-target scoring function. The R_p trend in the horizontal test is reported as a function of train set dimension, for the MLP scoring function created using experimental and synthetic data. As already said, the horizontal test presents an overestimation of the scoring function performance. However, the MLP per-target scoring function in the case of OX2 and JAK2 is similar or better in terms of R_p score than the performance of MLP scoring function in the horizontal test.

As a further consideration, all the per-target scoring functions show an R_p score increasing with the train set dimension. It seems that this increasing trend would be maintained if the database was increased even further. This suggests that the performance of the per-target scoring function can improve with larger databases.

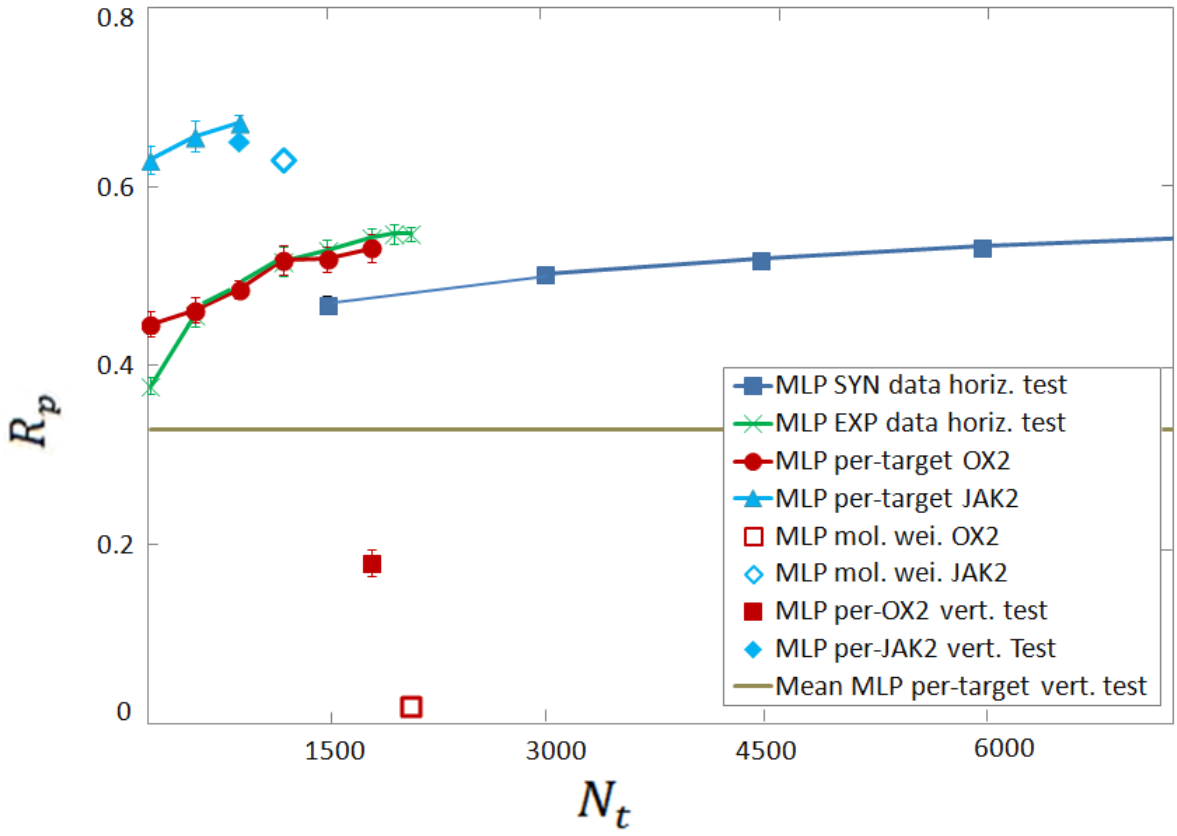

Figure 6.2 Comparison among the MLP per-target scoring function, the MLP scoring function in a per-protein vertical test, and a simple molecular weight scoring function The Pearson correlation coefficient $R_p$is plotted as a function of the training set size (x axes). The test set used in the comparison is homogeneous in terms of size and in terms of target. The considered target value is the ligand-protein. The horizontal line represents the mean Pearson correlation coefficient $R_p$obtained by the MLP scoring function in the per-target vertical test.

7 Conclusions

7.1 Research summary

As initially specified, the objective of the present study is to verify the real efficiency and the effective performances of the recently developed machine learning scoring functions. Here we investigated the circumstances in which a machine learning scoring function produces overestimated performances and why this happens. As a possible solution we proposed a tests protocol to be followed in order to guarantee a real performance description of machine learning scoring functions. Eventually an effective and innovative solution in the field of machine learning scoring functions was proposed.

In order to reach the aims of this research we moved toward the target through many steps. We collected experimental and synthetic 3D ligand-protein structures and we adjusted and corrected them in order to be suitable for training a machine learning scoring function. We studied different types of training protocols and network structures in order to increase as much as possible the scoring power of our scoring function. We proposed different types of data representation and we conducted an in-depth study on which is the most effective descriptive model to extract information from the data. We considered various types of possible tests to measure the performances of a machine learning scoring function, including, in particular, horizontal and vertical tests. We tried to give a clear picture of them related to the test type performed and to the possible real scenario reproduced with the particular test type. In addition we correlated the performances to the ones of types of scoring function previously developed in the field (both classic and machine learning)

7.2 Conclusions

The present research confirms that optimal results can be obtained in ligand-protein affinity prediction if horizontal tests are performed with machine learning scoring function, as previous studies have already suggested (96) (100) (103) (102) (98). The

correlation between predicted and real data is moderately strong to strong in all the scoring function presented in this thesis. As reported in previous studies (96) (100) (102) (101) (98), classic scoring functions reach lower level of correlation than those.

However, horizontal tests describe a particular utilizing case of a scoring function and are not indicative of its general performance. Indeed, a horizontal test puts a machine learning scoring function in a favorable situation respect to its effective capability of predicting binding affinity. In fact, as already discussed (118), machine learning scoring functions are able to distinguish different proteins if they undergo a training with a dataset including the same proteins used in test set. We confirm that machine learning scoring function learn to recognize similar proteins and their performance can depend on the percentage of database seen in the training phase, if the same proteins are present in the test set too. In particular, fixing the size of the database, the higher is the variance of the proteins in the database, the lower the performance of the machine learning scoring function is, if a horizontal test is performed. According to this observation, the comparison between classic scoring functions and machine learning scoring function using a horizontal test is not appropriate.

In addition, in the field of chemistry and pharmacy, the typical use of a scoring function is to help the discovery of active ligands for protein not yet studied, or for which a poor quantity of data is present. This scenario is described by a vertical test.
In this type of test, the MLP scoring function shows a degradation of performance in terms of R_p of 0.2. The performance degradation is confirmed also by other researchers who performed this type of test (98), or tests where the similarity between complexes present in train and test is avoided (100) (101). Also Jincai Yang *et al* (118) highlighted this situation in their research. Considering a vertical test, as underlined by Jose Jimenez *et al.* study (103), the degradation of the machine learning scoring power closes the gap between machine learning scoring functions performances and classic ones. Even the simple molecular weight scoring functions can guarantee a correlation between real and predicted data comparable to machine learning scoring function in this situation.
Because of this the performance of new machine learning scoring function should always be described by both horizontal and vertical tests. Only the contemporary presence of these tests can provide an exhaustive picture of the performance of the function. In the vertical test the training of the machine learning scoring function should be done on complexes different from the ones used for testing. To clearly establish what

avoiding similar complexes in train and test set means, the instruction of our study is to avoid the presence of same proteins contemporarily in test and train set. This is an objective and recognizable indication.

Eventually, the innovative solution proposed by this study in the field of machine learning scoring function is the use of machine learning per-target scoring function. This type of function requires (if not present yet) a small amount of previous work for experimentally measuring ligand-protein affinity for thousands of complexes of the interested protein. The data are used as training set in the recursive scheme presented with MPL per-protein scoring function. In the end, one obtains a scoring function which is expected to be more reliable and to guarantee at least a moderate correlation between real and predicted data. As presented, a similar level of performance is not guaranteed in vertical test by all the functions who performed it. To verify exactly the scoring power of a per-target scoring function, one can always use the small database created for training it, letting a small portion of data to test the function. In this way one achieves the awareness that the performance obtained are realistic or underestimated, if you own more data to increase your training database. In fact, one of the most interesting aspect of the machine learning per-target scoring function is the data used to create it. Synthetic data are used. This type of data are relative fast to be produced. In fact the complex structure is created using popular docking software and only the ligand-protein affinity is experimental measured. This is a relative fast procedure. For sure it is very quicker than producing a complete experimental data for which a 3D crystallography is necessary. This is why many more synthetic data are now available with respect to experimental data. The easiness in collecting synthetic data permits the availability of large synthetic database in which are more probable to find interesting target. On the other hand the per-target scoring function demonstrate to have improvable performance if larger database would be used, as the performance plot maintain a positive and constant gradient. Eventually the per-target scoring functions are now able to guarantee at least a moderate correlation between real and predicted data also with target of interest and this level of performance can be increased using larger database. Moreover, a per-target scoring function can be easily created also with descriptive models and neural network types presented by previous studies on the subject, just modifying the data used for training the machine learning scoring function.

Acknowledgements

I would like to thank all the team who participate in this work. First of all professor Sebastiano Pilati, since he drove my research activities and supervised all my work, professor Andrea Perali, the coordinator of the team, and professor Diego Dal Ben.
I would like to thank the department of Physics of Università di Camerino which host my research activity and the School of Advanced Studies of Università di Camerino and the whole Università di Camerino for the opportunity of attending the PhD course.

Bibliography

1. **MITCHELL, Tom M.** *Machine learning.* s.l. : McGraw-hill, 1997.

2. **MITCHELL, Tom, et al.** *Machine learning.* s.l. : Annual review of computer science, 1990. 4.1: 417-433.

3. **WANG, H., et al.** *Machine learning basics.* s.l. : Deep learning, 2016. 98-164.

4. **BARR, Avron, FEIGENBAUM, Edward A. and COHEN, Paul R.** *The handbook of artificial intelligence.* s.l. : William Kaufmann, 1981.

5. **WINSTON, Patrick Henry.** *Artificial intelligence.* s.l. : Addison-Wesley Longman Publishing Co., 1984.

6. **BUZKO, I., et al.** *Artificial Intelligence technologies in human resource development.* s.l. : Computer modelling and new technologies, 2016. 20.2: 26-29.

7. **EVANS, Guy-Warwick.** *Artificial intelligence: where we came from, where we are now, and where we are going.* 2017.

8. **PIAS, Claus.** *Analog, digital, and the cybernetic illusion.* s.l. : Kybernetes, 2005.

9. **STEELE, Guy.** *Common LISP: the language.* s.l. : Elsevier, 1990.

10. **CLOCKSIN, William F. and MELLISH, Christopher S.** *Programming in PROLOG.* s.l. : Springer Science & Business Media, 2003.

11. **ROSENBLATT, Frank.** *The perceptron: a probabilistic model for information storage and organization in the brain.* s.l. : Psychological review, 1958. 65.6: 386.

12. **MINSKY, Marvin and PAPERT, Seymour A.** *Perceptrons, Reissue of the 1988 Expanded Edition with a new foreword by Léon Bottou: An Introduction to Computational Geometry.* s.l. : MIT press, 2017.

13. **BRODTKORB, André R., HAGEN, Trond R. and SÆTRA, Martin L.** *Graphics processing unit (GPU) programming strategies and trends in GPU computing.* s.l. : Journal of Parallel and Distributed Computing, 2013. 73.1: 4-13.

14. **MARKOVIĆ, Danijela, et al.** *Physics for neuromorphic computing.* s.l. : Nature Reviews Physics, 2020. 2.9: 499-510.

15. **WEISS, Eric A.** *Biographies: Eloge: Arthur Lee Samuel (1901-90).* s.l. : IEEE Annals of the History of Computing, 1992. 14.3: 55-69.

16. **JORDAN, Michael I. and MITCHELL, Tom M.** *Machine learning: Trends, perspectives, and prospects.* s.l. : Science, 2015. 349.6245: 255-260.

17. **MAHESH, Batta.** *Machine learning algorithms-a review.* s.l. : International Journal of Science and Research (IJSR), 2020. 9: 381-386.

18. **AYODELE, Taiwo Oladipupo.** *AYODELE, Taiwo Oladipupo.* s.l. : New advances in machine learning, 2010. 3: 19-48.

19. **GENTLEMAN, Robert and CAREY, Vincent J.** *Unsupervised machine learning.* s.l. : Springer, 2008.

20. **SINGH, Amanpreet, THAKUR, Narina and SHARMA, Aakanksha.** *A review of supervised machine learning algorithms.* s.l. : Ieee, 2016.

21. **KAELBLING, Leslie Pack, LITTMAN, Michael L. and MOORE, Andrew W.** *Reinforcement learning: A survey.* s.l. : Journal of artificial intelligence research, 1996. 4: 237-285.

22. **PUTERMAN, Martin L.** *Markov decision processes.* s.l. : Handbooks in operations research and management science, 1990. 2: 331-434.

23. **MAĆKIEWICZ, Andrzej and RATAJCZAK, Waldemar.** *Principal components analysis (PCA).* s.l. : Computers & Geosciences, 1993. 19.3: 303-342.

24. **MADHULATHA, T. Soni.** *An overview on clustering methods.* s.l. : arXiv preprint arXiv, 2012. 1205.1117,.

25. **BIAU, Gérard and SCORNET, Erwan.** *A random forest guided tour.* s.l. : Test, 2016. 25.2: 197-227.

26. **DRAPER, Norman R. and SMITH, Harry.** *Applied regression analysis.* s.l. : John Wiley & Sons, 1998.

27. **NOBLE, William S.** *What is a support vector machine?* s.l. : Nature biotechnology, 2006. 24.12: 1565-1567.

28. **WANG, Sun-Chong.** *Artificial neural network.* s.l. : Interdisciplinary computing in java programming, 2003.

29. **SHARMA, Sagar, SHARMA, Simone and ATHAIYA, Anidhya.** *Activation functions in neural networks.* s.l. : towards data science, 2017. 6.12: 310-316.

30. **RUDER, Sebastian.** *An overview of gradient descent optimization algorithms.* s.l. : arXiv preprint arXiv, 2016. 1609.04747.

31. **CILIMKOVIC, Mirza.** *Neural networks and back propagation algorithm.* s.l. : Institute of Technology Blanchardstown, 2015. 15.1.

32. **JABBAR, H. and KHAN, Rafiqul Zaman.** *Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study).* s.l. : Computer Science, Communication and Instrumentation Devices, 2015. 70.

33. **GIROSI, Federico, JONES, Michael and POGGIO, Tomaso.** *Regularization theory and neural networks architectures.* 1995. Vol. Neural computation. 7.2: 219-269.

34. **VLACHOS, Andreas.** *A stopping criterion for active learning.* s.l. : Computer Speech & Language, 2008. 22.3: 295-312.

35. **TORREY, Lisa and SHAVLIK, Jude.** *Transfer learning.* s.l. : Handbook of research on machine learning applications and trends: algorithms, methods, and techniques. IGI global, 2010. p. 242-264.

36. **ZHUANG, Fuzhen, et al.** *A comprehensive survey on transfer learning.* s.l. : Proceedings of the IEEE, 2020. 109.1: 43-76.

37. **AGRAWAL, Divyakant, et al.** *Challenges and opportunities with Big Data.* 2011. 2011-1..

38. **RUSSOM, Philip, et al.** *Big data analytics.* 2011. Vols. TDWI best practices report, fourth quarter. 19.4: 1-34..

39. **WHITFORD, David.** *Proteins: structure and function.* s.l. : John Wiley & Sons, 2013.

40. **MEISTER, Alton.** *Biochemistry of the amino acids.* s.l. : Elsevier, 2012.

41. **TALLAWI, Marwa, et al.** *Strategies for the chemical and biological functionalization of scaffolds for cardiac tissue engineering: a review.* s.l. : Journal of the Royal Society Interface, 2015. 12.108: 20150254.

42. **HANSEN, Jeffrey L., et al.** *Structural insights into peptide bond formation.* s.l. : Proceedings of the National Academy of Sciences, 2002. 99.18: 11670-11675.

43. **EISENHABER, Frank, PERSSON, Bengt and ARGOS, Patrick.** *Protein structure prediction: recognition of primary, secondary, and tertiary structural features from amino acid sequence.* s.l. : Critical reviews in biochemistry and molecular biology, 1995. 30.1: 1-94.

44. **REYNOLDS, Charles H., TOUNGE, Brett A. and BEMBENEK, Scott D.** *Ligand binding efficiency: trends, physical basis, and implications.* 2008. Vol. Journal of medicinal chemistry. 51.8: 2432-2438.

45. **HOMANS, S. W.** *Dynamics and thermodynamics of ligand–protein interactions.* s.l. : Bioactive Conformation I, 2006. 51-82.

46. **BERMAN, Helen M.** *The protein data bank: a historical perspective.* s.l. : Acta Crystallographica Section A, 2008. 64.1: 88-95.

47. **LASKOWSKI, Roman A., et al.** *PDBsum: a Web-based database of summaries and analyses of all PDB structures.* s.l. : Trends in biochemical sciences, 1997. 22.12: 488-490.

48. **MEYER, Edgar F.** *The first years of the Protein Data Bank.* s.l. : Protein science: a publication of the Protein Society, 1997. 6.7: 1591.

49. **WANG, R. Fang. X., et al., et al.** *The PDBbind Database: Methodologies and Updates.* s.l. : J. Med. Chem, 2005. 48: 4111-4119.

50. **WANG, Renxiao, et al.** *The PDBbind database: Collection of binding affinities for protein− ligand complexes with known three-dimensional structures.* s.l. : Journal of medicinal chemistry, 2004. 47.12: 2977-2980.

51. **DUNBAR JR, James B., et al.** *CSAR data set release 2012: ligands, affinities, complexes, and docking decoys.* s.l. : Journal of chemical information and modeling, 2013. 53.8: 1842-1852.

52. **GESTWICKI, Jason, et al.** *CSAR data set release 2012: Ligands, affinities, complexes, and docking decoys.* 2013.

53. **GILSON, Michael K. and ZHOU, Huan-Xiang.** *Calculation of protein-ligand binding affinities.* s.l. : Annual review of biophysics and biomolecular structure, 2007. 36.1: 21-42.

54. **MERZ JR, Kenneth M., RINGE, Dagmar and REYNOLDS, Charles H.** *Drug design: structure-and ligand-based approaches.* s.l. : Cambridge University Press, 2010.

55. **GILSON, Michael K., et al.** *BindingDB in 2015: a public database for medicinal chemistry, computational chemistry and systems pharmacology.* s.l. : Nucleic acids research, 2016. 44.D1: D1045-D1053.

56. **TROTT, Oleg and OLSON, Arthur J.** *AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading.* s.l. : Journal of computational chemistry, 2010. 31.2: 455-461.

57. **WANG, Lingle, et al.** *Accurate and reliable prediction of relative ligand binding potency in prospective drug discovery by way of a modern free-energy calculation protocol and force field.* s.l. : Journal of the American Chemical Society, 2015. 137.7: 2695-2703.

58. **DRAGIEV, Plamen, NADON, Robert and MAKARENKOV, Vladimir.** *Systematic error detection in experimental high-throughput screening.* s.l. : BMC bioinformatics, 2011. 12.1: 1-14..

59. **JONES, Gareth, et al.** *Development and validation of a genetic algorithm for flexible docking.* s.l. : Journal of molecular biology, 1997. 267.3: 727-748.

60. **JAIN, Ajay N.** *Surflex: fully automatic flexible molecular docking using a molecular similarity-based search engine.* s.l. : ournal of medicinal chemistry, 2003. 46.4: 499-511.

61. **BÖHM, Hans-Joachim.** *Prediction of binding constants of protein ligands: a fast method for the prioritization of hits obtained from de novo design or 3D database search programs.* s.l. : Journal of computer-aided molecular design, 1998. 12.4: 309-309.

62. **HUANG, Niu, et al.** *Molecular mechanics methods for predicting protein–ligand binding.* s.l. : Physical Chemistry Chemical Physics, 2006. 8.44: 5166-5177.

63. **KITCHEN, Douglas B., et al.** *Docking and scoring in virtual screening for drug discovery: methods and applications.* s.l. : Nature reviews Drug discovery, 2004. 3.11: 935-949..

64. **MUEGGE, Ingo.** *PMF scoring revisited.* s.l. : Journal of medicinal chemistry, 2006. 49.20: 5895-5902.

65. **KIRTAY, Chrysi Konstantinou, MITCHELL, John BO and LUMLEY, James A.** *Knowledge based potentials: The reverse Boltzmann methodology, virtual screening and molecular weight dependence.* s.l. : QSAR & Combinatorial Science, 2005. 24.4: 527-536.

66. **MUEGGE, Ingo and MARTIN, Yvonne C.** *A general and fast scoring function for protein− ligand interactions: a simplified potential approach.* s.l. : Journal of medicinal chemistry, 1999. 42.5: 791-804.

67. **GOHLKE, Holger, HENDLICH, Manfred and KLEBE, Gerhard.** *Knowledge-based scoring function to predict protein-ligand interactions.* s.l. : Journal of molecular biology, 2000. 295.2: 337-356.

68. **VELEC, Hans FG, GOHLKE, Holger and KLEBE, Gerhard.** *DrugScoreCSD knowledge-based scoring function derived from small molecule crystal data with superior recognition rate of near-native ligand poses and better affinity prediction.* s.l. : Journal of medicinal chemistry, 2005. 48.20: 6296-6303.

69. **KULHARIA, Mahesh, GOODY, Roger S. and JACKSON, Richard M.** *Information Theory-Based Scoring Function for the Structure-Based Prediction of Protein− Ligand Binding Affinity.* s.l. : Journal of chemical information and modeling, 2008. 48.10: 1990-1998.

70. **GUVENCH, Olgun and MACKERELL JR, Alexander D.** *Computational fragment-based binding site identification by ligand competitive saturation.* s.l. : PLoS computational biology, 2009. 5.7: e1000435..

71. **WAN, Shunzhou, et al.** *Evaluation and characterization of Trk kinase inhibitors for the treatment of pain: reliable binding affinity predictions from theory and computation.* s.l. : Journal of Chemical Information and Modeling, 2017. 57.4: 897-909.

72. **CIORDIA, Myriam, et al.** *Application of free energy perturbation for the design of BACE1 inhibitors.* s.l. : Journal of Chemical information and modeling, 2016. 56.9: 1856-1871.

73. **KERANEN, Henrik, et al.** *Acylguanidine beta secretase 1 inhibitors: a combined experimental and free energy perturbation study.* s.l. : Journal of chemical theory and computation, 2017. 13.3: 1439-1453.

74. **GILSON, Michael K. and ZHOU, Huan-Xiang.** *Calculation of protein-ligand binding affinities.* s.l. : Annual review of biophysics and biomolecular structure, 2007. 36.1: 21-42.

75. **LEACH, Andrew R.** *Molecular modelling: principles and applications.* s.l. : Pearson education, 2001.

76. **IRWIN, John J.** *Community benchmarks for virtual screening.* s.l. : Journal of computer-aided molecular design, 2008. 22.3: 193-199.

77. **SCHNEIDER, Gisbert.** *Virtual screening: an endless staircase?* s.l. : Nature Reviews Drug Discovery, 2010. 9.4: 273-276.

78. **SPYRAKIS, Francesca, et al.** *The consequences of scoring docked ligand conformations using free energy correlations.* s.l. : European journal of medicinal chemistry, 2007. 42.7: 921-933.

79. **LI, Hongjian, et al.** *Improving AutoDock Vina using random forest: the growing accuracy of binding affinity prediction by the effective exploitation of larger data sets.* s.l. : Molecular informatics, 2015. 34.2-3: 115-126.

80. **AIN, Qurrat Ul, et al.** *Machine-learning scoring functions to improve structure-based binding affinity prediction and virtual screening.* s.l. : Wiley Interdisciplinary Reviews: Computational Molecular Science, 2015. 5.6: 405-424.

81. **CHENG, Tiejun, et al.** *Structure-based virtual screening for drug discovery: a problem-centric review.* s.l. : The AAPS journal, 2012. 14.1: 133-141.

82. **HUANG, Sheng-You, GRINTER, Sam Z. and ZOU, Xiaoqin.** *Scoring functions and their evaluation methods for protein–ligand docking: recent advances and future directions.* s.l. : Physical Chemistry Chemical Physics, 2010. 12.40: 12899-12908.

83. **MA, Dik-Lung, CHAN, Daniel Shiu-Hin and LEUNG, Chung-Hang.** *Drug repositioning by structure-based virtual screening.* s.l. : Chemical Society Reviews, 2013. 42.5: 2130-2141.

84. **ASHTAWY, Hossam M. and MAHAPATRA, Nihar R.** *A comparative assessment of predictive accuracies of conventional and machine learning scoring functions for protein-ligand binding affinity prediction.* s.l. : EEE/ACM transactions on computational biology and bioinformatics, 2014. 12.2: 335-347.

85. **LI, Hongjian, et al.** *Substituting random forest for multiple linear regression improves binding affinity prediction of scoring functions: Cyscore as a case study.* s.l. : BMC bioinformatics, 2014. 15.1: 1-12.

86. **BALLESTER, Pedro J., SCHREYER, Adrian and BLUNDELL, Tom L.** *Does a more precise chemical description of protein–ligand complexes lead to more accurate prediction of binding affinity?* s.l. : Journal of chemical information and modeling, Journal of chemical information and modeling. 54.3: 944-955.

87. **BALLESTER, Pedro J. and MITCHELL, John BO.** *Comments on "leave-cluster-out cross-validation is appropriate for scoring functions derived from diverse protein data sets": Significance for the validation of scoring functions.* s.l. : Journal of chemical information and modeling, 2011. 51.8: 1739-1741.

88. **LI, Hongjian, et al.** *Low-quality structural and interaction data improves binding affinity prediction via random forest.* s.l. : Molecules, 2015. 20.6: 10947-10962.

89. —. *Correcting the impact of docking pose generation error on binding affinity prediction.* s.l. : BMC bioinformatics, 2016. 17.11: 13-25.

90. **PIRES, Douglas EV and ASCHER, David B.** *CSM-lig: a web server for assessing and comparing protein–small molecule affinities.* s.l. : Nucleic acids research, 2016. 44.W1: W557-W561.

91. **WÓJCIKOWSKI, Maciej, ZIELENKIEWICZ, Piotr and SIEDLECKI, Pawel.** *Open Drug Discovery Toolkit (ODDT): a new open-source player in the drug discovery field. .* s.l. : Journal of cheminformatics, 2015. 7.1: 1-6.

92. **DENG, Wei, BRENEMAN, Curt and EMBRECHTS, Mark J.** *Predicting protein− ligand binding affinities using novel geometrical descriptors and machine-learning methods.* s.l. : Journal of chemical information and computer sciences, 2004. 4.2: 699-703.

93. **AMINI, Ata, et al.** *A general approach for developing system-specific functions to score protein–ligand docked complexes using support vector inductive logic programming.* s.l. : roteins: Structure, Function, and Bioinformatics, 2007. 69.4: 823-831.

94. **WANG, Renxiao, LU, Yipin and WANG, Shaomeng.** *Comparative evaluation of 11 scoring functions for molecular docking.* s.l. : Journal of medicinal chemistry, 2003. 46.12: 2287-2303.

95. **SOTRIFFER, Christoph A., et al.** *SFCscore: scoring functions for affinity prediction of protein–ligand complexes.* s.l. : Proteins: Structure, Function, and Bioinformatics, 2008. 73.2: 395-419.

96. **BALLESTER, Pedro J. and MITCHELL, John BO.** *A machine learning approach to predicting protein–ligand binding affinity with applications to molecular docking.* s.l. : Bioinformatics, 2010. 26.9: 1169-1175.

97. **BREIMAN, Leo.** *Random forests.* s.l. : Machine learning, 2001. 45.1: 5-32.

98. **WÓJCIKOWSKI, Maciej, BALLESTER, Pedro J. and SIEDLECKI, Pawel.** *Performance of machine-learning scoring functions in structure-based virtual screening.* s.l. : Scientific Reports, 2017. 7.1: 1-10.

99. **KRIZHEVSKY, Alex, SUTSKEVER, Ilya and HINTON, Geoffrey E.** *Imagenet classification with deep convolutional neural networks.* s.l. : Advances in neural information processing systems, 2012. 25.

100. **GOMES, Joseph, et al.** *Atomic convolutional networks for predicting protein-ligand binding affinity.* s.l. : rXiv preprint arXiv, 2017. 1703.10603.

101. **SEO, Sangmin, et al.** *Binding affinity prediction for protein–ligand complex using deep attention mechanism based on intermolecular interactions.* s.l. : BMC bioinformatics, 2021. 22.1: 1-15.

102. **STEPNIEWSKA-DZIUBINSKA, Marta M., ZIELENKIEWICZ, Piotr and SIEDLECKI, Pawel.** *Development and evaluation of a deep learning model for protein–ligand binding affinity prediction. .* s.l. : Bioinformatics, 2018. 34.21: 3666-3674.

103. **JIMÉNEZ, José, et al.** *K deep: protein–ligand absolute binding affinity prediction via 3d-convolutional neural networks.* s.l. : Journal of chemical information and modeling, 2018. 58.2: 287-296.

104. **IANDOLA, Forrest N., et al.** *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size.* s.l. : arXiv preprint arXiv, 2016. 1602.07360.

105. **DUNBAR JR, James B., et al.** *DUNBAR JR, James B., et al. CSAR benchmark exercise of 2010: selection of the protein–ligand complexes.* s.l. : Journal of chemical information and modeling, 2011. 51.9: 2036-2046.

106. **LIU, Zhihai, et al.** *Forging the basis for developing protein–ligand interaction scoring functions.* s.l. : Accounts of chemical research, 2017. 50.2: 302-309.

107. **HUANG, Niu, SHOICHET, Brian K. and IRWIN, John J.** *Benchmarking sets for molecular docking.* s.l. : Journal of medicinal chemistry, 2006. 49.23: 6789-6801.

108. **MYSINGER, Michael M., et al.** *Directory of useful decoys, enhanced (DUD-E): better ligands and decoys for better benchmarking.* s.l. : Journal of medicinal chemistry, 2012. 55.14: 6582-6594.

109. **HUANG, Niu, SHOICHET, Brian K. and IRWIN, John J.** *Benchmarking sets for molecular docking.* s.l. : Journal of medicinal chemistry, 2006. 49.23: 6789-6801.

110. **KOES, David Ryan, BAUMGARTNER, Matthew P. and CAMACHO, Carlos J.** *Lessons learned in empirical scoring with smina from the CSAR 2011 benchmarking exercise.* s.l. : Journal of chemical information and modeling, 2013. 53.8: 1893-1904.

111. **MYSINGER, Michael M. and SHOICHET, Brian K.** *Rapid context-dependent ligand desolvation in molecular docking.* s.l. : Journal of chemical information and modeling, 2010. 50.9: 1561-1573.

112. **KUNTZ, Irwin D., et al.** *A geometric approach to macromolecule-ligand interactions.* s.l. : Journal of molecular biology, 1982. 161.2: 269-288.

113. **LANG, P. Therese, et al.** *DOCK 6: Combining techniques to model RNA–small molecule complexes.* s.l. : Rna, 2009. 15.6: 1219-1230.

114. **HARTSHORN, Michael J., et al.** *Diverse, high-quality test set for the validation of protein− ligand docking performance.* s.l. : Journal of medicinal chemistry, 2007. 50.4: 726-741.

115. **LI, Yang and YANG, Jianyi.** *Structural and sequence similarity makes a significant impact on machine-learning-based scoring functions for protein–ligand interactions.* s.l. : Journal of chemical information and modeling, 2017. 57.4: 1007-1012.

116. **WÓJCIKOWSKI, M. Kukie lka, M., Stepniewska-Dziubinska, MM and Siedlecki, P.** *Development of a protein–ligand extended connectivity (PLEC) fingerprint and its application for binding affinity predictions.* s.l. : Bioinformatics, 2018.

117. **ZHENG, Liangzhen, FAN, Jingrong and MU, Yuguang.** *Onionnet: a multiple-layer intermolecular-contact-based convolutional neural network for protein–ligand binding affinity prediction.* s.l. : ACS omega, 2019. 4.14: 15956-15965.

118. **YANG, Jincai, SHEN, Cheng and HUANG, Niu.** *Predicting or pretending: artificial intelligence for protein-ligand interactions lack of sufficiently large and unbiased datasets.* s.l. : Frontiers in pharmacology, 2020. 11: 69.

119. **John Goerzen, Brandon Rhodes.** *Foundations of Python Network Programming: The Comprehensive Guide to Building Network Applications with Python.* 2010. 978-1-4302-3003-8.

120. **ROSENBLATT, Frank.** *Principles of neurodynamics. perceptrons and the theory of brain mechanisms.* s.l. : Cornell Aeronautical Lab Inc Buffalo NY, 1961.

121. **CYBENKO, George.** *Approximation by superpositions of a sigmoidal function.* s.l. : Mathematics of control, signals and systems, 1989. 2.4: 303-314.

122. **KINGMA, Diederik P. and BA, Jimmy.** *Adam: A method for stochastic optimization.* s.l. : arXiv preprint arXiv, 2014. 1412.6980.

123. **Dragiev, P., Nadon, R., & Makarenkov, V.** *Systematic error detection in experimental high-throughput screening.* s.l. : BMC bioinformatics, 2011.